

LifeChain: A Hybrid Cross-Chain Blockchain Framework for Interoperable and Integrity-Preserving Organ Donation and Transplant Matching

Khalida Tabassum, Syed Imtiyaz Hassan

Department of Computer Science and Information Technology, Maulana Azad National Urdu University (Central University),
Hyderabad, 500032, India

Abstract—Organ allocation depends on timely and trustworthy exchange of donor and recipient information among hospitals that run independent information systems. Existing blockchain solutions for healthcare usually rely on a single ledger, which forces a trade-off between the confidentiality required for donor data and the public auditability required for allocation decisions. To resolve this trade-off, this study proposes LifeChain, a hybrid cross-chain framework that keeps donor records on a permissioned Hyperledger Fabric ledger, publishes organ requests and match confirmations on a public Ethereum network, and links the two through a Hyperledger Cactus interoperability gateway. Every cross-chain relay carries a Keccak-256 digest and an Elliptic Curve Digital Signature Algorithm (ECDSA) signature of the gateway, which the Ethereum contract verifies on-chain; the contract thus authenticates the gateway's attestation, while the correctness of the attested Fabric state rests on the gateway, a trust assumption that the study makes explicit. On a single-host prototype with a two-organization Fabric network and the Ethereum Sepolia testnet, under light load, the private ledger processed 153.9 and 154.3 transactions per second (TPS) in a baseline run and a 100-transaction stress run; the public contract interface processed 20.8 and 20.5 TPS and the Cactus bridge 57.8 and 58.8 TPS; these are observed rates, not capacities. Off-chain processing of a cross-chain match took 251.9 ms (baseline run), excluding public block inclusion, which adds at least one 12 s Sepolia slot. All altered and forged relays submitted by the security audit were rejected, and the digests matched on both ledgers for every confirmed match. Compared with a homogeneous Fabric-to-Fabric federation, LifeChain adds public verifiability across heterogeneous ledgers at the cost of a confirmation time dominated by public-chain inclusion.

Keywords—Blockchain interoperability; cross-chain communication; Hyperledger Fabric; Hyperledger Cactus; Ethereum; data integrity; organ donation

I. INTRODUCTION

Organ transplantation is one of the most time-sensitive workflows in clinical workflow. Following the identification of a deceased or living donor, the two hospitals - the donor and recipient hospitals- together with the allocating authority have to agree on the compatibility, priority, and logistics within a narrow window of viability, and all decisions have to be subject to regulatory and legal audit at a later stage [1]. In reality, these organizations operate their own hospital information systems, and allocation continues to be done via registry exports, phone confirmation, and manually reconciling spreadsheets. This is a

type of problem that is difficult to audit, slow, and susceptible to record manipulation, all for which distributed ledgers were proposed [2], [3].

Blockchain-based healthcare systems have come a long way, from ledgers for access control like MedRec [4] and Ancile [5] to FHIR-focused solutions for data sharing [6] and focused platforms for organ donation [7]. However, most of these systems are constructed upon a single ledger. One permissioned ledger ensures donor privacy, but does not have an independent public audit trail, and one public ledger ensures transparency, but is not appropriate for personally identifiable clinical data as per regulations like the General Data Protection Regulation (GDPR) [8]. Both are required for organ allocation: donor information should be kept within a group of accredited hospitals, and the presence of a request, match, and fulfillment should be verifiable by everyone.

Blockchain interoperability offers a way out of this dilemma [9], [10]. In recent work, federations of health ledgers [11] have been connected, chains of heterogeneous blockchain chains in supply-chain applications [12] have been connected, and public and private ledgers have been connected for credential verification in hiring [13]. The studies demonstrate that it is possible to design and implement cross-chain solutions, but they either link chains of the same type, consider only functional behaviour without taking into account the end-to-end timing, or don't include a cryptographic locking of the payload they are relaying to the receiving contract. For a safety-critical domain like organ allocation, the relay itself should be verifiable, and its latency layer by layer will be measured.

For this reason, this study introduces a hybrid cross-chain system called LifeChain, which stores donor information on a private Hyperledger Fabric ledger [14], simulates organ requests and match confirmations on a public Ethereum network [15], and integrates them with a Hyperledger Cactus gateway [16]. The proposed work will contribute in the following ways:

- We propose a generalized, chain-agnostic framework for interoperable and integrity-preserving communication between heterogeneous blockchains, with a canonical cross-chain message model, a seven-phase protocol in which every message is hashed, signed, and verified on-chain by the destination, and an anchoring invariant that makes each exchange auditable on both ledgers (Section III).

- We apply the framework to organ donation as LifeChain, which keeps confidential donor state on Fabric, publishes allocation events on Ethereum, and couples them through a Hyperledger Cactus gateway (Section IV).
- We analyse the security of the proposed framework under an explicit adversary model, with proof sketches that reduce relay forgery to the unforgeability of ECDSA and the second-preimage resistance of Keccak-256, state the trust placed in the gateway, and quantify the on-chain cost of verification (Section V).
- We provide a full implementation, including a two-organization Fabric network, a Solidity contract deployed and verified on the Sepolia testnet, a scoring-based matching engine, PostgreSQL indexing, and a hospital portal with four-stage organ transport tracking (Section VI).
- We report a layer-wise performance and security evaluation of the prototype under light load, covering observed throughput, off-chain processing latency per stage, public-chain inclusion and finality, container resource use, an analytical bottleneck bound, gas cost, tamper rejection over a defined attack set, and identity-uniqueness enforcement (Section VII).
- We compare the proposed framework with three existing interoperability works: a Fabric federation for electronic health records [11], heterogeneous supply-chain blockchains [12], and the HIREX hiring platform [13], distinguishing values reported in those works from values computed or measured in this study (Section VIII).

The remainder of the study is organized as follows. Section II reviews related work. Section III presents the generalized interoperability framework, and Section IV applies it to the LifeChain organ donation system. Section V analyses security, Section VI details the implementation, Section VII reports the experimental results, Section VIII compares the proposed work with existing works, Section IX discusses limitations, and Section X concludes the study.

II. RELATED WORK

A. Blockchain in Healthcare Data Management

Kuo *et al.* [3] identified decentralized management, immutable audit trails, and data provenance as the main benefits of distributed ledgers in biomedicine. MedRec [4] used Ethereum smart contracts to manage access permissions to records stored off-chain, and Ancile [5] extended this model with role-based access control and proxy re-encryption. FHIRChain team [6] integrated Health Level Seven (HL7) Fast Healthcare Interoperability Resources (FHIR) with a token-based access control mechanism for clinical data sharing. The ledgers used in these systems illustrate the importance of accountability, but each of these systems relies on a single ledger, thus picking up its confidentiality and/or transparency constraints.

B. Blockchain for Organ Donation and Transplantation

Hawashin *et al.* [7] proposed an Ethereum-based organ donation and transplantation system in which smart contracts register donors and recipients, run matching, and track the transplant process. All the participant data is stored on one public-style ledger, privacy is guaranteed by off-chain storage, and their work introduces the concept of allocation logic that is automated and auditable. The proposed LifeChain framework keeps donor identity and clinical attributes completely within the private consortium and only exposes anonymized identifiers, scores, and digests on the public chain.

C. Blockchain Interoperability

Belchior *et al.* [9] break interoperability solutions down into public connectors (sidechains, relays, and hash-locking), blockchain-of-blockchains platforms like Cosmos [17] and Polkadot [18], and hybrid connectors that feature trusted relays and blockchain-agnostic protocols. Time-locked contracts are hash-based contracts that can be used to do atomic swaps between chains, but they are not used for the communication of application state. Enterprise-oriented approaches instead rely on relays with verifiable proofs [20] or on transaction-based cross-chain invocation [21]. Hyperledger Cactus [16] follows the hybrid-connector model: ledger-specific connector plug-ins are orchestrated by a gateway that can execute and verify cross-ledger business logic.

D. Target Interoperability Studies

Three recent works are directly relevant and serve as the existing baselines against which the proposed framework is compared in Section VIII. Hashim *et al.* [11] interconnected two permissioned Hyperledger Fabric consortia for sharing electronic health records. Local and global smart contracts separate intra- and inter-blockchain transactions; Hyperledger Cactus carries the inter-blockchain calls, and a hierarchical public-key infrastructure (PKI) links the certificate authorities of both federations. Using Hyperledger Caliper on two Intel Core i5 machines, they reported about 496 transactions per second at a 500 transactions-per-second send rate, average latencies of 0.868 s and 0.922 s for the two networks, and a steady-state inter-blockchain transfer time of 1.0125 s, compared with about 3.05 s for an earlier trusted-execution-based relay. Their design is, however, homogeneous: both sides are Fabric networks, and no public ledger is involved.

Echtermeyer [12] studied interoperability between heterogeneous blockchains in a supply-chain setting, where trading partners operate different ledger platforms and product provenance must be followed across them. The work highlights the practical obstacles of heterogeneous integration, such as differing data models, consensus guarantees, and identity schemes, which are also present when a permissioned health ledger is connected to a public network.

Amin *et al.* have proposed a heterogeneous interoperable blockchain solution for the hiring process called HIREX [13]. Academic or experience certificates are awarded on the blockchain of the issuing entity, such as Ethereum, Hyperledger Fabric, and Quorum, or a Node.js service layer that is built on top of a React front end allows applicants to build verifiable credentials from multiple chains together in one application.

HIREX demonstrates heterogeneous interoperability in a credential-verification context, with an evaluation that focuses on functional behaviour.

E. Research Gap

The reviewed studies leave three gaps that are addressed by the proposed LifeChain framework. First, public-private heterogeneity is very rarely coupled with a confidentiality barrier where confidential data is all on the private side. Second, on-chain signature checks for relayed payloads are rare, meaning that relay integrity relies on trust in the relayer. Third, the end-to-end cross-chain latency is typically presented as one total unit without breaking it down by layer to pinpoint latency issues.

III. PROPOSED GENERALIZED FRAMEWORK FOR INTEROPERABLE AND INTEGRITY-PRESERVING CROSS-CHAIN COMMUNICATION

In this section, the main contribution of the study is introduced: a generalized, chain-agnostic approach to state and event exchange between heterogeneous blockchain systems, while maintaining the integrity of each exchanged message. The framework is not tied to any specific application domain, and it is then illustrated as an application for organ donation called the LifeChain in Section IV.

A. Importance of Interoperability and Integrity

Various differences exist between blockchain platforms in terms of consensus protocol, membership model, data model, smart-contract language, and identity scheme [9]. Permissioned ledgers like Hyperledger Fabric support high throughput and finely-grained access control while maintaining confidentiality for a known consortium [14] while permissionless ledgers, like Ethereum, offer an open state with independent verification [15]. Real processes don't always belong to a single ledger – they cross organisations with different platforms, and they require confidential and public records. Each of these ledgers is an isolated silo unless they are interconnected, meaning all the data has to be manually re-entered or passed to a centralized middleware, reintroducing the single point of trust which blockchains were designed to eliminate and disrupting the end-to-end audit trail [10] [21].

Interoperability alone, however, is not sufficient. A cross-chain message leaves the security domain of the source consensus as soon as it is read from the source ledger. Between the source and the destination, it passes through connectors, relayers, and networks, where it can be altered, forged, replayed, or reordered. The consensus of the destination ledger only guarantees that whatever is submitted is recorded immutably; it does not guarantee that the submitted content is what the source ledger actually committed. Cross-chain bridges have consequently become a prominent attack surface of multi-ledger systems [9]. *Integrity* therefore has to be an explicit property of the communication itself: a destination ledger must accept a cross-chain message only if it can verify that the message is authentic, unmodified, fresh, and authorized. The proposed framework treats interoperability and integrity as two inseparable requirements:

- R1 – Interoperability: Any ledger B_i can consume events and invoke operations of any other ledger B_j , regardless of their consensus, membership or data model.
- R2 – Integrity: A message is accepted on the destination ledger only if it is the same as the state committed on the source ledger and was passed via an authorized gateway – and the act of accepting is recorded on-chain.
- R3 – Confidentiality by placement: The data that should remain confidential does NOT cross the boundary, only pseudonymous identifiers and digests.
- R4 – Auditability: The cross-chain exchange should create matching and verifiable records on both ledgers.
- R5 – Low overhead: Integrity verification incurs a small, bounded cost, both in terms of latency and fees.

B. System and Threat Model

Let $\mathcal{B} = \{B_1, \dots, B_n\}$ be a set of heterogeneous ledgers. Each ledger B_i has its own consensus, state S_i , and identity scheme, and is either private (permissioned) or public (permissionless). An interoperability gateway G holds a signing key pair (sk_G, pk_G) and one connector C_i per ledger. Every destination ledger hosts a small *verifier contract* that knows pk_G (or the derived address $addr_G$) and keeps a registry $\mathcal{N}_{dst}$ of consumed message nonces. Domain applications interact with the ledgers through an orchestration layer.

The adversary may observe all public transactions, may attempt to register duplicate or fake identities to gain influence (a Sybil attack [22]), may alter, drop, reorder, or replay messages in transit between the ledgers, and may try to forge relay messages without the gateway key. We assume that the hash function is second-preimage resistant, that the signature scheme is existentially unforgeable, that the consensus of each ledger is sound (for example, at most a minority of Raft orderers fail by crashing [23]), and that sk_G is protected by the operator. Denial-of-service attacks and compromise of end-user devices are outside the scope. The gateway is therefore a trusted attester: the framework guarantees that a destination accepts only what the gateway signed, not that what the gateway signed is a correct reading of the source ledger. Section V states this assumption formally and analyses the consequences of a compromised gateway, and Section IX discusses how to distribute the trust.

C. Layered Architecture of the Framework

Fig. 1 shows the four layers. The application layer (L1) hosts domain decentralized applications (dApps) and audit tools. The orchestration layer (L2) contains the domain business logic, listens for ledger events, and may keep a rebuildable off-chain index for fast queries. The heterogeneous ledger layer (L3) contains any combination of private and public ledgers. The interoperability and integrity layer (L4) is realized by the gateway and comprises five components:

Ledger-specific connectors that read events and submit transactions.

A message builder that assembles a canonical cross-chain message.

An integrity engine that hashes and signs the message.

Verifier contracts on each destination that check the message before it can change state.

An anchoring and audit component that writes the result back to the source ledger.

Adding a new ledger requires only one connector and one verifier contract, which makes the framework extensible to any platform that supports hashing and signature verification in its smart contracts.

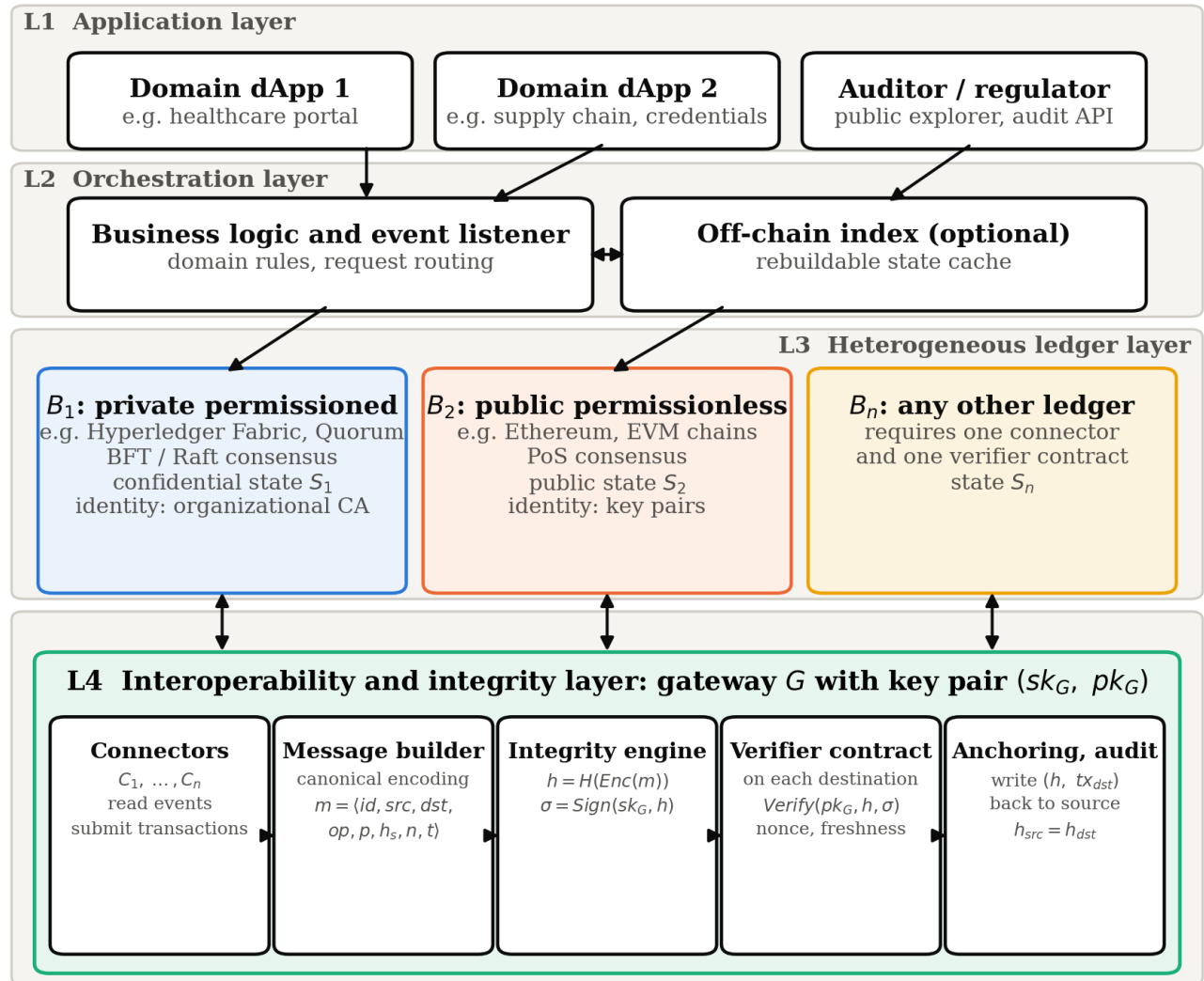


Fig. 1. Proposed generalized framework for interoperable and integrity-preserving communication between heterogeneous blockchains: domain applications (L1), orchestration (L2), heterogeneous ledgers (L3) and the interoperability and integrity layer (L4) with connectors, message builder, integrity engine, verifier contracts and anchoring.

D. Cross-Chain Message Model

Every exchange between two ledgers is expressed as a cross-chain message.

$$m = \langle id, src, dst, op, p, h_s, n, t \rangle \quad (1)$$

where, id is a unique message identifier, src and dst are the source and destination ledgers, op is the requested operation, p is the application payload, h_s is a digest of the source state on which the message is based, n is a nonce, and t is a timestamp. The message builder serializes m with a deterministic canonical encoding $Enc(\cdot)$, so that the source, the gateway and the destination compute identical bytes for identical content. $Enc(\cdot)$ must be injective: every field is encoded with a fixed type and

width in a fixed order, and variable-length fields are first hashed to a fixed-length value, so that two different messages can never produce the same byte string. Section IV-F gives the concrete encoding used for Ethereum.

E. Integrity-Preserving Communication Protocol

The protocol has seven phases.

- 1) *Event capture*: the connector of the source ledger observes an event that requires a cross-chain action.
- 2) *State extraction*: the gateway reads the relevant source state and computes its digest h_s .
- 3) *Digest computation*: the integrity engine computes

$$h = H(\text{Enc}(m)) \quad (2)$$

4) *Signing*: the gateway signs the digest,

$$\sigma = \text{Sign}(\text{sk}_G, h) \quad (3)$$

5) *Submission*: the connector of the destination ledger submits (m, σ) .

6) *On-chain verification*: the verifier contract recomputes h from the submitted fields and accepts the message only if all three conditions hold:

$$\text{Verify}(\text{pk}_G, h, \sigma) = 1 \quad (4)$$

$$n \notin \mathcal{N}_{\text{dst}}, \quad t_{\text{now}} - t \leq \Delta \quad (5)$$

that is, the signature is valid for the recomputed digest, the nonce has not been consumed before, and the timestamp lies inside the freshness window: the message is not older than Δ , and it is not dated more than a small clock-skew allowance ε in the future ($t \leq t_{\text{now}} + \varepsilon$), so that a future-dated message cannot extend its own validity. A message that repeats a consumed nonce is rejected as a duplicate. On acceptance, n is added to $\mathcal{N}_{\text{dst}}$ and the state change is committed.

7) *Anchoring*: the gateway writes $(h, \text{tx}_{\text{dst}})$ back to the source ledger, where tx_{dst} is the destination transaction hash, so that both ledgers hold the same fingerprint of the exchange:

$$h^{(\text{src})} = h^{(\text{dst})} \quad (6)$$

The difference between the two stored digests is an auditable integrity invariant of the framework: any difference detected later implies tampering with the framework. The protocol is summarized in a generic form in Algorithm 1.

Algorithm 1. Generic integrity-preserving cross-chain communication

Input: event e on source ledger B_s ; destination B_d ; gateway keys $(\text{sk}_G, \text{pk}_G)$
Output: verified state change on B_d and anchor on B_s , or rejection
1: $x \leftarrow C_s.\text{readState}(e)$; $h_s \leftarrow H(x)$
2: $m \leftarrow \langle \text{id}, s, d, \text{op}, p(x), h_s, n, t \rangle$
3: $h \leftarrow H(\text{Enc}(m))$; $\sigma \leftarrow \text{Sign}(\text{sk}_G, h)$
4: $\text{tx} \leftarrow C_d.\text{submit}(m, \sigma)$
5: $\triangleright$ verifier contract on B_d :
6: **require** $\text{Verify}(\text{pk}_G, H(\text{Enc}(m)), \sigma) = 1$
7: **require** $n \notin \mathcal{N}_{\text{dst}}$ **and** $t \leq t_{\text{now}} + \varepsilon$ **and** $t_{\text{now}} - t \leq \Delta$
8: $\mathcal{N}_{\text{dst}} \leftarrow \mathcal{N}_{\text{dst}} \cup \{n\}$; apply $\text{op}(p)$
9: if tx reverted **then** log REJECTED; **return**
10: $C_s.\text{anchor}(\text{id}, h, \text{tx}.\text{hash}) \quad \triangleright$ Eq. (6)

F. Performance Model

An end-to-end latency of one cross-chain exchange is broken down into the latency of the source ledger, the gateway (hashing, signing, and relay), and the destination ledger and optional index.

$$L_{\text{E2E}} = L_{\text{src}} + L_G + L_{\text{dst}} + L_{\text{idx}} \quad (7)$$

The slowest of these stages restricts and bounds the sustainable exchange rate. This decomposition enables localization of bottlenecks in a specific layer, instead of on the cross-chain path as a whole.

G. Design Properties

Table I maps each requirement to its mechanism. Because the destination verifies every message itself, trust in the relay path is replaced by verification, which is the central difference between the proposed framework and relay designs that forward messages without an on-chain check.

TABLE I. REQUIREMENTS OF THE GENERALIZED FRAMEWORK AND THE MECHANISMS THAT SATISFY THEM

Requirement	Mechanism in the framework	Verified by
R1 Interoperability	Connectors per ledger; canonical message m ; verifier contract per destination	Section IV instantiation
R2 Integrity	Digest [Eq. (2)], signature [Eq. (3)], on-chain checks [Eq. (4), (5)]	Section V; tamper-rejection and integrity rates (Section VII)
R3 Confidentiality	Only identifiers and digests cross ledger boundaries	Data placement analysis
R4 Auditability	Anchoring invariant [Eq. (6)] on both ledgers	Digest parity tests
R5 Low overhead	Constant-size digest and one signature check per message	Latency and gas results

IV. APPLICATION OF THE PROPOSED FRAMEWORK: LIFECHAIN ORGAN DONATION SYSTEM

The conundrum of organ donation is that all five requirements must be satisfied: donor information must be confidential (R3), allocation must be made under a system of public audit (R4), the allocation must not be falsified or corrupted (R2), hospitals have different systems in place (R1), and the allocation must be carried out in time (R5). Table II shows how each generic component is instantiated in LifeChain.

TABLE II. INSTANTIATION OF THE GENERALIZED FRAMEWORK IN LIFECHAIN

Generic component	LifeChain instantiation
Private ledger B_1	Hyperledger Fabric v2.x, channel mychannel, donor registry chaincode
Public ledger B_2	Ethereum Sepolia, Solidity LifeChain contract
Gateway G and connectors	Hyperledger Cactus with Fabric and Ethereum connector plug-ins
Message m	Match relay (request ID, donor ID, score S , h_F , t)
Hash H	Keccak-256 for relay digests; SHA-256 for Fabric record fingerprints
Sign / Verify	secp256k1 ECDSA / ecrecover against addr_G
Nonce registry $\mathcal{N}_{\text{dst}}$	Request ID, consumed once by the PENDING $\rightarrow$ CONFIRMED transition; it's checked against window Δ , ε (Section IV-F)
Anchoring	Fabric recordMatch(h , Ethereum tx hash)
Application/orchestration	React hospital portal; Node.js API, matching engine, PostgreSQL index

A. Design Goals and Methodology

Following the requirements R1–R5, LifeChain is designed around five goals derived from the research gap: (DG1) personal donor data never leaves the permissioned consortium; (DG2) every allocation decision is publicly verifiable; (DG3) a relayed

match cannot be forged or altered without detection; (DG4) each hospital has exactly one on-chain identity; and (DG5) the off-chain part of the cross-chain path (gateway, Fabric and contract call, excluding public block inclusion) adds sub-second processing latency. The proposed method involves 6 steps: 1) Hospital registration on both ledgers, 2) Donor registration on Fabric, 3) Requesting the publishing on Ethereum, 4) Cross-chain compatibility search and scoring by the gateway, 5) Signed relay and on-chain verification of the compatibility, and 6) Signed transport-phase tracking until transplantation. The participants, the architecture, the placement of data, the matching model, and the integrity protocol are described in the following subsections [19].

B. Participants

There are 4 roles in the application. Hospitals are accredited hospitals that register donors, raise organ requests, and arrange for transport; each hospital has a Fabric identity issued by its certificate authority and an Ethereum account. The *interoperability gateway* operates the Cactus connectors and holds the relayer signing key sk_G . *Auditors and regulators* read the public ledger and the transaction history. The *network operator* manages the Fabric ordering service and chaincode life cycle.

The threat model of Section III applies unchanged, with hospitals as the identities that an adversary may try to duplicate and relays as the messages that it may try to alter, forge or replay.

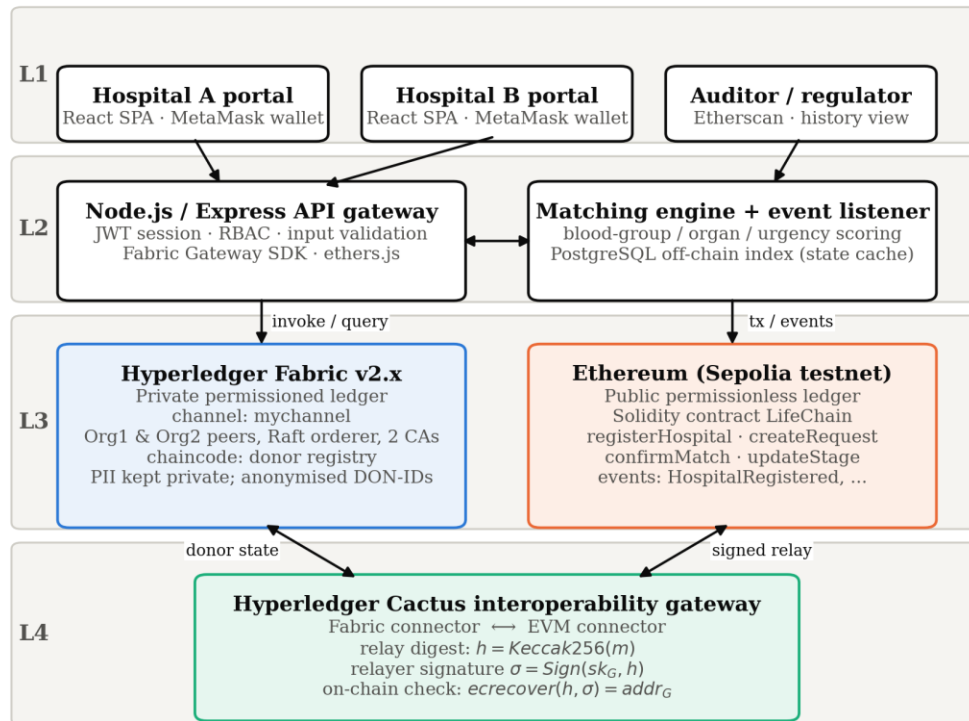


Fig. 2. LifeChain architecture as an instance of the generalized framework: L1 presentation, L2 application and service, L3 heterogeneous ledgers (private Hyperledger Fabric and public Ethereum Sepolia) and L4 Hyperledger Cactus interoperability and integrity layer.

C. LifeChain Architecture

Fig. 2 shows the LifeChain instance of the four-layer framework. The *presentation layer* (L1) is a React single-page application through which hospital staff register donors, raise requests, review matches, and update transport stages; auditors additionally use the Etherscan explorer. The application and service layer (L2) is a Node.js/Express API that authenticates users and validates input, calls Fabric from the Fabric Gateway SDK, and submits Ethereum transactions via ethers.js over JSON remote procedure calls (JSON-RPC). There are two layers: one with a matching engine, one with an event listener, and a PostgreSQL database indexes ledger state to avoid scanning either chain for dashboard queries. The ledger layer (L3) is intentionally multi-pronged. The permissioned Fabric network (channel mychannel, two peer organizations, a Raft ordering service, and one certificate authority per organization)

stores donor records. Other elements only see an anonymized identifier, like DON-DD7115, the organ type, the ABO blood group, and a digest of the private record; personally identifiable information (PII) is never accessed. The public Ethereum contract keeps details about hospital registrations, requests for organs, confirmations of a match, the progress of the organ being transported, and emits events like OrganRegistered for off-chain listeners. The *interoperability and integrity layer* (L4) is the Cactus gateway, which uses a Fabric connector and an Ethereum connector to read state from one ledger and commit verified results to the other.

D. Data Placement and Privacy by Design

Table III shows where each data item of the proposed framework is stored. Only pseudonymous identifiers, compatibility attributes, and 32-byte digests are stored on Fabric, the public chain; the PostgreSQL index stores both a

pseudonymous and a rebuildable cache. No items on the public ledger contain any identification of a person; the clinical record connects to it via the salted fingerprint h_F , which is included in the call data of the match confirmation. This satisfies DG1 while still meeting DG2; the residual risks of publishing h_F and event metadata are analysed in Proposition 4.

TABLE III. DATA PLACEMENT IN THE PROPOSED FRAMEWORK

Data item	Fabric (private)	Ethereum (public)	Index (cache)
Donor identity and clinical record (PII)	Yes	No	No
Record fingerprint h_F (SHA-256)	Yes	Yes (calldata of confirmMatch; salted)	No
Anonymized donor ID (DON-xxxxxx)	Yes	Yes	Yes
Organ type and ABO group	Yes	Yes	Yes
Hospital ID, name, admin address	No	Yes	Yes
Request ID and urgency	No	Yes	Yes
Match score S	Yes	Yes	Yes
Relay digest h and signature σ	h only	Yes	No
Transport stage updates	No	Yes	Yes
Ethereum transaction hash	Yes	Native	Yes

E. Data Model and Matching

A donor record d on Fabric contains the anonymized identifier, the donating hospital, the organ type o_d , the blood group b_d , a registration time, and the digest $H(\text{PII})$ of the confidential attributes. A request r on Ethereum contains the request identifier, the requesting hospital, the organ type o_r , the recipient blood group b_r , and an urgency level u_r . Organ type is a hard constraint, and ABO compatibility follows the standard donor–recipient rule set:

$$c(d, r) = \begin{cases} 1, & o_d = o_r \wedge b_d \in \text{Comp}(b_r) \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

where, $\text{Comp}(b_r)$ is the set of ABO groups that can donate to b_r . Among compatible donors, the matching engine ranks candidates with a normalized score

$$S(d, r) = 100 c(d, r) [w_1 m_b + w_2 u_r + w_3 f_d] \quad (9)$$

with $w_1 + w_2 + w_3 = 1$ and $w_i \geq 0$, so that $S \in [0, 100]$. Every component is normalized to $[0, 1]$ before weighting: $m_b(d, r)$ equals 1 for an identical ABO group and a lower value for a compatible but non-identical group; u_r maps the ordinal urgency levels of a request onto $[0, 1]$; and f_d decreases from 1 to 0 as the time since donor registration approaches the viability window of the organ. The weights and the component mappings are configuration parameters of the matching engine, and all scores reported in Section VII (100 % for REQ040, 84 % and 87 % for REQ032 and REQ036) were produced with the default configuration of the prototype. These defaults were set by the authors to exercise the protocol; they were not calibrated against clinical outcomes or elicited from clinicians, and Eq. (9) is not proposed as an allocation policy. The following two properties,

which are confirmed by the functional tests, were used in this study for validation: organ type and ABO compatibility are hard filters that no weighting can overcome; and the score is monotone in each of its components, as can be seen by construction since each weighting is non-negative. The cross-chain protocol lets S be an opaque, signed field, making it possible to replace Eq. (9) with an existing allocation rule, such as a points-based national kidney allocation policy [29] or the criteria of the competent authority under [1]. Section IX discusses the requirements of such a rule.

F. Instantiated Cross-Chain Matching Protocol

Algorithm 1 is implemented as LifeChain. On receiving a request event emitted by the Ethereum contract, the gateway asks Fabric for a list of donors that are suitable to the request, calculates the score and constructs a canonical relay payload, which contains the request identifier, the donor identifier, the score, the Fabric state digest h_F (the SHA-256 fingerprint [24] of the donor record in Fabric), and a timestamp t . It then derives

$$h = \text{Keccak256}(\text{id}_r \parallel \text{id}_d \parallel S \parallel h_F \parallel t) \quad (10)$$

where, id_r and id_d are the request and donor identifiers, and $\parallel$ denotes concatenation. This is Eq. (2) with $H = \text{Keccak-256}$ [25]; the signature of Eq. (3) is realized with secp256k1 ECDSA [26], and the verification of Eq. (4) is realized with the ecrecover precompile. The contract function that confirms a match recomputes h from the submitted fields, recovers the signer address with ecrecover , and rejects the transaction unless the address equals the registered gateway address addr_G . The same digest, along with the Ethereum transaction hash, is written back to Fabric, and both ledgers contain the same fingerprint for each match, while the record fingerprint (SHA-256) maintains the compatibility of the private ledger with standard Fabric tooling. Any change to the payload in transit alters h and causes any message to be invalid; if h is altered through a change to the payload, then any message signed with a key other than sk_G fails the address check. The procedure is summarized in Algorithm 2.

1) *Hashed data and canonical serialization*: LifeChain specifies that the five fields of Eq. (10) are encoded with the Solidity ABI encoding $\text{abi.encode}(\text{bytes32 id}_r, \text{bytes32 id}_d, \text{uint256 } S, \text{bytes32 } h_F, \text{uint256 } t)$, in which every field occupies exactly one 32-byte word in a fixed order; the request and donor identifier strings are first mapped to bytes32 with Keccak-256 over their UTF-8 bytes, S is an integer percentage and t is a Unix time in seconds. The gateway computes the same encoding off-chain with the ethers.js ABI coder, and the contract recomputes it on-chain, so both sides hash an identical 160-byte input; this is also the five-word input assumed in the gas analysis of Table V. Because every field has a fixed width, the encoding is injective, which the $\parallel$ notation of Eq. (10) abbreviates. On the private side, $h_F = \text{SHA-256}(\text{salt}_d \parallel \text{Enc}_F(d))$, where Enc_F serializes the donor record as JSON with lexicographically sorted keys, UTF-8 encoding and no insignificant whitespace, and salt_d is a 256-bit random value generated at registration and stored only on Fabric.

2) *What the contract verifies*: The contract checks that the five fields were signed by the registered gateway and that the

request is pending. It does not, and on Ethereum alone cannot, verify that h_F is the current state of a Fabric donor record, that the donor exists, or that S was computed correctly: on Ethereum, h_F is an opaque commitment. Its correspondence with Fabric state can be checked only on the Fabric side, at anchoring: the recordMatch chaincode function is specified to recompute the fingerprint of the stored donor record and to endorse the anchor only if it equals the h_F carried in the relay, so that a mismatch is recorded as an integrity alarm instead of an anchor. Cross-chain integrity in LifeChain therefore means relay integrity together with digest parity: the public ledger holds exactly what the gateway attested, and the private ledger confirms that the attestation refers to its own state. It is not a trustless proof of Fabric state on Ethereum; the resulting trust in the gateway is analysed in Proposition 6 and discussed in Section IX.

3) *Nonces, freshness and duplicates*: LifeChain does not keep a separate nonce registry. The role of the nonce n of Eq. (1) is played by the request identifier id_r , which the contract allows to be consumed exactly once: confirmMatch requires the request to be PENDING and moves it to CONFIRMED in the same transaction. A duplicate or replayed confirmation for the same request therefore reverts, whether it repeats the original signature or names a different donor, and two confirmations for the same request cannot both be included because the second

one executes against the updated state. The timestamp t is bound into h , so it cannot be changed without invalidating σ , and it is stored with the match for audit. Freshness is specified as $\text{require}(t \leq \text{block.timestamp} + \varepsilon)$ and $\text{require}(\text{block.timestamp} - t \leq \Delta)$, where the window Δ and the clock-skew allowance ε are consortium parameters; these checks reject both stale and future-dated relays. Even without them, replay protection holds through the one-time state transition, and t then serves as an audit timestamp.

Algorithm 2. LifeChain cross-chain match and verified relay (instance of Algorithm 1)

Input: request event r from Ethereum; gateway key sk_G

Output: confirmed match on both ledgers, or rejection

1: $D \leftarrow \text{Fabric.queryCompatibleDonors}(o_r, b_r)$

2: if $D = \emptyset$ then keep r in state PENDING; **return**

3: $d^* \leftarrow \text{argmax}_{d \in D} S(d, r) \quad \triangleright \text{Eq. (9)}$

4: $h_F \leftarrow \text{Fabric.getStateHash}(d^*)$

5: $h \leftarrow \text{Keccak256}(r.\text{id} \parallel d^*.\text{id} \parallel S \parallel h_F \parallel t)$

6: $\sigma \leftarrow \text{ECDSA.Sign}(sk_G, h)$

7: $\text{tx} \leftarrow \text{Ethereum.confirmMatch}(r.\text{id}, d^*.\text{id}, S, h_F, t, \sigma)$

8: $\triangleright$ on-chain: $\text{require ecrecover}(h, \sigma) = \text{addr}_G \wedge \text{state}(r) = \text{PENDING} \wedge \text{freshness (Section IV-F)}$

9: if tx reverted then log TAMPER_REJECTED; **return**

10: $\text{Fabric.recordMatch}(d^*.\text{id}, r.\text{id}, h, \text{tx.hash})$

11: $\text{PostgreSQL.index}(r, d^*, S, \text{tx.hash})$; notify hospitals

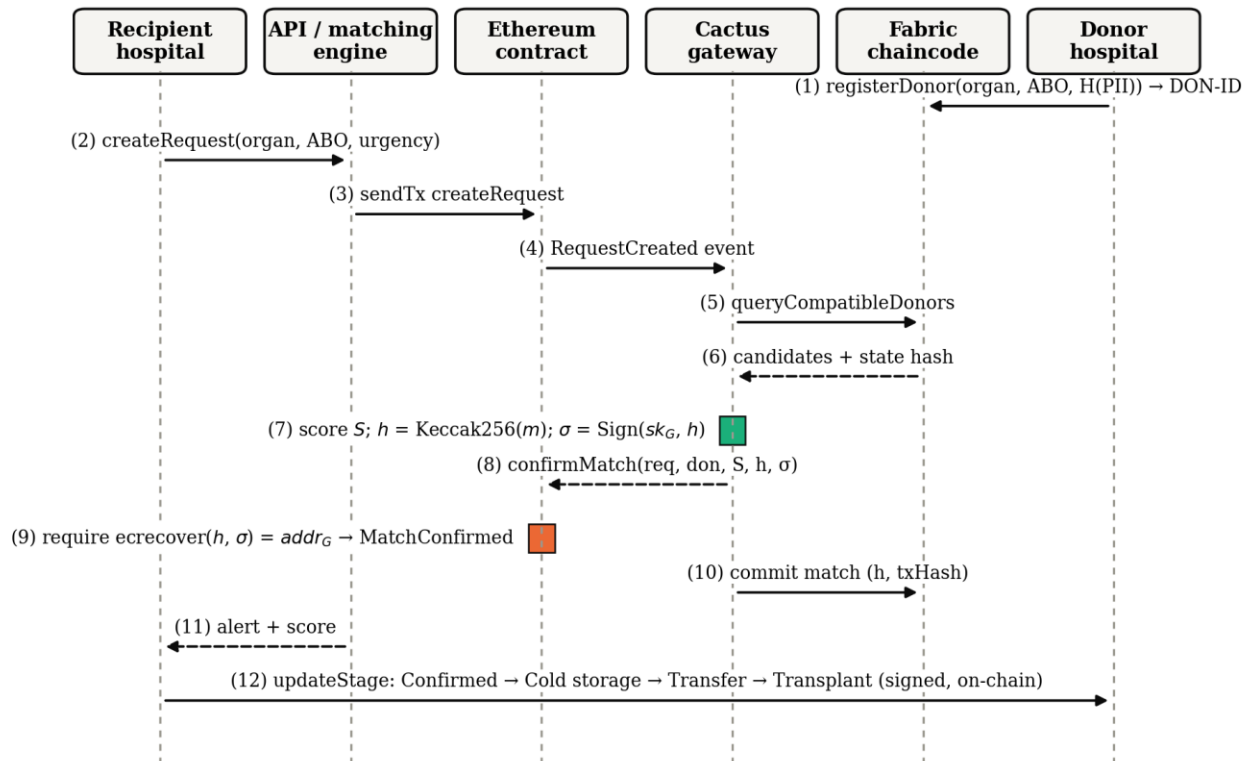


Fig. 3. Message sequence of a cross-chain donor match, from donor registration on Fabric to signed transport-stage updates. Dashed arrows denote responses; filled markers denote local computation at the gateway and on-chain verification.

G. Transport Provenance and Identity Integrity

Once confirmed, the organ goes through 4 stages: Match Confirmed, In Cold Storage, Hospital Transfer, and Transplant. Every update is signed by the hospital that makes it and committed on-chain to create a tamper-proof cold-chain record. Each hospital identifier is unique in the contract, and if a second registration is made using the same identifier, the registration process rolls back with the message “Hospital ID already registered”, thus ensuring that one operator cannot have multiple IDs to inflate its “allocation power” [22]. The entire message flow is depicted in Fig. 3.

V. SECURITY ANALYSIS OF THE PROPOSED FRAMEWORK

A. Security Properties

This section compares the proposed framework with the threat model of Section III. The propositions below are stated with proof sketches under the adversary model that follows; they are not machine-checked proofs. The secp256k1 ECDSA scheme offers about 128-bit security [26], and Keccak-256 offers 256-bit second-preimage and 128-bit collision resistance [25].

Adversary model. The adversary $\mathcal{A}$ is a probabilistic polynomial-time algorithm that controls the network between the gateway and both ledgers: it may read, drop, delay, reorder, replay, and inject messages. It may submit arbitrary transactions to Ethereum from accounts it controls, register hospitals under identifiers of its choice, and observe every relay that the honest gateway signs during normal operation, which gives it an adaptive chosen-message view of the gateway's signatures. If $\mathcal{A}$ sees a relay-integrity match (id_r, id_d, S, h_F, t) that the honest gateway did not sign then $\mathcal{A}$ wins the relay-integrity game. Proposition 6 assumes that there is no compromise of the gateway and the gateway is not covered by this game.

1) *Proposition 1 (relay integrity):* Without skG on an adversary, a match can be accepted by the contract only if it is a valid ECDSA signature on a fresh digest or if an adversary can find another preimage of an existing digest. Proof sketch: the contract recalculates h from the fields that it receives [Eq. (10)], and only accepts the input if it satisfies Eq. (4). Any update to any field in the preimage causes h to differ unless a second preimage is found, and then a new h necessitates a new signature to be verified under addrG . The winning $\mathcal{A}$ therefore either outputs a valid signature on a digest that the gateway never signed, which is a violation of the existential unforgeability of ECDSA under adaptive chosen-message attack, or outputs fields different from a signed tuple with the same digest, which is a second preimage of Keccak-256 for the injective encoding of Section IV-F. Hence $\Pr[\mathcal{A} \text{ wins}] \leq \text{Adv}_{\text{ECDSA}}^{\text{EUF-CMA}}(\mathcal{A}) + \text{Adv}_{\text{Keccak}}^{\text{SPR}}(\mathcal{A})$, which is of order 2^{-128} per attempt and negligible. The tamper-rejection results of Section VII demonstrate that the checks are in effect and are accessible, and demonstrate that the guarantee depends on this reduction, rather than on the finite test set.

2) *Proposition 2 (replay resistance):* If a match is captured, it cannot be replayed to create another match. Proof sketch: The request identifier is sent as the nonce and is included in h , which

means a confirmation can only be sent for one particular request. The contract only accepts a confirmation for a request in the PENDING state and changes its state to CONFIRMED in the same transaction; a replay of that request, or a duplicate with the same set of confirmation data, will be executed against the CONFIRMED state and will be reverted. The timestamp t is locked in h and compared to the window $[t_{\text{now}} - \Delta, t_{\text{now}} + \epsilon]$.

3) *Proposition 3:* Each accepted match will be tied to a public transaction hash, as well as the gateway address, and Fabric will keep the public transaction hash along with that h . Both the gateway and a hospital can not deny a match and/or a transport-stage update later, as they are signed by their issuer and recorded on an append-only ledger [14, 15].

4) *Proposition 4 (confidentiality):* There is no personally identifiable information directly visible on the public ledger, and a salted h_F is immune to dictionary attacks, but event metadata allows linkages to be made. In the public chain, the pseudonymous identifiers, organ type, ABO group, urgency, score, timestamps, and h_F are kept by construction (Table III). Since the type and ABO group of an organ donor are public, an unsalted fingerprint could be vulnerable to a dictionary attack, in which $\mathcal{A}$ searches for potential organ donors, computes h_D , and compares it to h_F . To make h_F a hiding commitment, the salt is a 256-bit random salt value, which can only be kept by Fabric (Section IV-F), and LifeChain computes the h_F over this salt together with the record. However, hashing can be linked by metadata. The times of requests, matches, and transport stages on the public chain, combined with the organ type and the hospital, might be correlated with external information, such as the news of a death or an admission, to re-identify a donor or recipient. There is an inherent risk of this because coarse-grained timestamps or batching events reduce but do not eliminate this risk, and this is one of the reasons why pseudonymized allocation events should be considered personal data (Section IX). 5)

5) *Proposition 5 (identity uniqueness):* Each hospital identifier is registered to a single on-chain entity, as a second registration with the same identifier would revert, and the donor-side operations also involve a Fabric identity from an organizational certificate authority. Thus, a single operator will not be able to multiply its identities to get a bigger share of allocations without the consent of a consortium certificate authority [22].

6) *Proposition 6 (trust in the gateway):* If skG is breached, any match accepted on Ethereum may be a fake, taken from an individual that does not exist, or h_F is misstated, or the score is incorrectly calculated. Only the signature of the gateway and the request state (Section IV-F) are verified in the contract, which means that possession of skG is enough. Preventing this inconsistency is easy, since recordMatch will not allow an h_F to be recorded that contains an inconsistency with a previously stored record, but it is not detected by any auditor who compares the two ledgers. The cross-chain integrity of LifeChain is thus dependent on one operator. To prevent this, the single trusted signer needs to be removed, such as by

verifying Fabric endorsement signatures within the contract (Section IX), or by having the threshold signature across gateways working for different consortium members (t-of-n).

Table IV summarizes the threats by the STRIDE categories (spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege), along with replay and Sybil attacks, and comprises the categories and the associated mitigation and evidence. The current design does not address the issues of a compromised gateway key, which is a separate trust assumption (Proposition 6).

TABLE IV. THREAT ANALYSIS OF THE PROPOSED FRAMEWORK (STRIDE)

Threat	Attack example	Mitigation in proposed LifeChain	Evidence
Spoofing	Fake relayer or hospital	ecrecover against $addr_G$; CA-issued Fabric identities	Prop. 1, 5; Table XIII
Tampering	Altered payload in transit	Keccak-256 digest recomputed in contract	Prop. 1; TRR = 100 %
Repudiation	Denying a match or stage update	Signed transactions on both ledgers	Prop. 3
Information disclosure	Reading donor data on public chain	PII only on Fabric; salted h_F ; pseudonymous IDs	Prop. 4; Table III
Denial of service	Flooding the gateway	Raft ordering, stateless gateway restart	Out of scope (Section IX)
Elevation of privilege	Unauthorized request or update	JSON Web Token (JWT) sessions, role-based access, endorsement policy	Functional tests
Replay	Resubmitting an old confirmation	One-time PENDING $\rightarrow$ CONFIRMED transition; t and request ID in h ; window Δ , ϵ	Prop. 2
Sybil	Duplicate hospital identities	Unique hospital ID in contract	TC4; Prop. 5

TABLE V. ANALYTICAL GAS COST OF RELAY VERIFICATION

Operation	Gas rule	Gas
Keccak-256 over 5 words	$30 + 6$ per word	60
ecrecover precompile	Fixed	3,000
Signature calldata (65 B)	≤ 16 per non-zero byte	$\leq 1,040$
Total verification overhead	–	$\approx 4,100$
Share of measured registration tx	4,100 / 168,805	2.4 %

B. Cost of on-Chain Verification

The security of the relay rests on two operations executed inside the contract: one Keccak-256 hash over the relay fields and one call to the `ecrecover` precompile. Using the Ethereum gas schedule [15], the incremental cost of verification can be computed analytically (Table V). Hashing five 32-byte words costs $30 + 6 \times 5 = 60$ gas, the precompile costs 3,000 gas, and the 65-byte signature adds at most $65 \times 16 = 1,040$ gas of calldata, for a total of about 4,100 gas. This is 2.4 % of the 168,805 gas measured for a hospital registration (Section VII) and about 19.5 % of the fixed 21,000-gas cost of any Ethereum

transaction, so the integrity guarantee of Proposition 1 is obtained at a small marginal cost.

VI. IMPLEMENTATION OF THE PROPOSED FRAMEWORK

A. Technology Stack and Testbed

Table VI lists the components of the prototype. The Fabric network runs as five Docker containers (two peers, one Raft orderer and two certificate authorities) under Docker on Windows Subsystem for Linux (WSL), with 8 GB of memory available to the containers. The Solidity contract is deployed on the Ethereum Sepolia testnet at address `0x490A0d25704133adDB6BD3E5a8899af64233C93` and is publicly inspectable on Etherscan. All components except the public chain ran on a single Windows workstation (Docker Desktop with the WSL 2 back end), and all experiments reported here were executed on 31 July 2026. Running every node on one host means that peers, orderer, gateway, database, and portal share one processor and one network stack; there is no network latency between Fabric nodes, and the results therefore characterize a single-host prototype, not a geographically distributed consortium.

The lower rows of Table VI give the experimental configuration. Each run was executed once, so the values in Section VII are means over the transactions of a single run and no run-to-run confidence intervals are available; Section IX lists this among the limitations.

TABLE VI. IMPLEMENTATION STACK AND TESTBED CONFIGURATION

Layer	Component	Technology/configuration
L1	Hospital portal	React SPA, Material UI, MetaMask wallet
L2	API gateway	Node.js / Express, JWT sessions, role-based access
L2	Ledger SDKs	Fabric Gateway SDK; ethers.js JSON-RPC client
L2	Off-chain index	PostgreSQL (state cache, dashboard queries)
L3	Private ledger	Hyperledger Fabric v2.x; channel mychannel; Org1, Org2 peers; Raft orderer; 2 CAs
L3	Public ledger	Ethereum Sepolia testnet; Solidity contract LifeChain
L4	Interoperability	Hyperledger Cactus gateway with Fabric and Ethereum connector plug-ins
L4	Integrity	Keccak-256 and SHA-256 digests; secp256k1 ECDSA; on-chain <code>ecrecover</code>
Host	Runtime	Single workstation; Docker Desktop (WSL 2); 8 GB container memory
L3	Fabric configuration	Fabric v2.x test-network settings; default block-cutting parameters; endorsement by the peers of Org1 and Org2
L3	Public network access	Sepolia through a public JSON-RPC endpoint over the Internet; one gateway account; gas price from the endpoint estimate
Bench	Workload	Baseline run under normal portal usage; stress run of 100 live transactions issued by the benchmark module; one run of each; per-stage means reported
Bench	Network conditions	Fabric containers on one Docker bridge network, no injected delay or loss; public side subject to Internet and endpoint conditions

B. Smart Contract and Chaincode

The Ethereum contract defines functions for hospital registration, creation of the request, confirmation of the match, handling updates while in the hospital transport phase, and includes one event for every state change. The registration event is `HospitalRegistered(uint256 indexed id, address indexed`

`admin, string name)`; which means the id of the hospital, the address of the admin who registers, and the name of the hospital are indexed and included in the data. The Fabric chaincode offers donor registration, compatibility queries, retrieval, and recording of state-hash. The logical interface is summarized in Table VII, where the names of operations are abbreviated for ease of presentation.

TABLE VII. LOGICAL INTERFACE OF THE CONTRACT AND CHAINCODE

Ledger	Operation/event	Purpose
Ethereum	registerHospital(id, name)	Onboard a hospital; reverts on duplicate identifier
Ethereum	createRequest(organ, ABO, urgency)	Publish an organ request (event RequestCreated)
Ethereum	confirmMatch(req, don, S, hF, t, σ)	Verify gateway signature with ecrecover; record match
Ethereum	updateStage(req, stage)	Append a signed transport-stage update
Ethereum	HospitalRegistered(id, admin, name)	Event consumed by the gateway and index
Fabric	registerDonor(organ, ABO, H(PII))	Store donor record; return anonymized DON-ID
Fabric	queryCompatibleDonors(organ, ABO)	Return candidates satisfying Eq. (8)
Fabric	recordMatch(don, req, h, txHash)	Store cross-ledger fingerprint of the match

C. Gateway, Matching Engine and Portal

The Cactus gateway registers for contract events via the Ethereum connector, invokes the chaincode via the Fabric connector, runs Algorithm 2, and returns the confirmation signed by it. The same service updates the PostgreSQL index, from which the dashboard analytics (requests, matches, participating hospitals, success rate, demand by organ type and by blood group) and the transaction history are updated. A built-in benchmark module runs batches of transactions and reports throughput, per-stage latency, and container statistics from the Docker engine.

D. Functional Workflow

The end-to-end workflow was exercised with several hospitals (City General Hospital, Samatha Hospital with identifier HOSP007, Hospital2 and Hospital3). A hospital administrator registers the institution, which triggers a

`HospitalRegistered` transaction on Sepolia, and then logs in with the hospital identifier. A donor (for instance, a heart valve from an O+ donor, anonymized as DON-0323BB) is registered on Fabric. When a request cannot be served immediately, it is recorded on-chain in a pending state and remains open until a compatible donor appears. Fig. 4 shows request REQ040 (heart, O+) matched through the gateway with donor DON-DD7115 at a score of 100 %. For kidney requests REQ032 and REQ036 (A-), the engine proposed donor DON-16D05F from Hospital2 with scores of 84 % and 87 %, and the portal exposes the donor hospital's contact details together with the four-stage transport tracker (Fig. 5). The history view in Fig. 6 links the three ledger actions of one allocation: the Fabric donor registration at 14:12:10, the Ethereum request at 14:13:03 and the Cactus match confirmation at 14:13:08. The complete set of functional test cases is reported in Section VII.

Create Organ Request

Organ

Heart

Blood Group

O+

Urgency

Medium

 **Match Found via Cactus Gateway**

Matched Donor: **DON-DD7115**

Score: 100%

SUBMIT REQUEST

Fig. 4. Organ request REQ040 (heart, O+, medium urgency) matched through the Cactus gateway with donor DON-DD7115 at a compatibility score of 100 %.

Hospital Contact Info

Matched Hospital

Hospital2

Hospital ID: HOSP002

Matched Donor Details

Donor ID: **DON-16D05F**

Organ: Kidney (A-)

Coordination Contacts

+1 (555) 019-2834

hosp002@hospital.org

Organ Transport Lifecycle

Update Stage: Match Confirmed

1 Match Confirmed

2 In Cold Transit

3 Hospital Received

4 Transplant Complete

CLOSE

Fig. 5. Multi-hospital coordination: donor DON-16D05F (kidney, A-) at Hospital2 matched to requests REQ032 and REQ036, with contact details and the four-stage organ transport lifecycle.

Request ID	Action	Blockchain Network	Date & Time
REQ040	Match Confirmed Matched with DON-DD7115 (Score: 100%)	Cactus Gateway	31/7/2026, 2:13:08 pm
REQ040	Organ Request Requested Heart (O+) – Medium	Ethereum	31/7/2026, 2:13:03 pm
DON-DD7115	Donor Registration Registered Heart (O+)	Hyperledger Fabric	31/7/2026, 2:12:10 pm

Fig. 6. Cross-chain transaction history for one allocation: donor registration on Hyperledger Fabric, organ request on Ethereum, and match confirmation through the Cactus gateway.

VII. RESULTS AND PERFORMANCE EVALUATION

A. Metrics and Methodology

Throughput is the number of committed transactions N_c divided by the time between the first and the last commit,

$$TPS = \frac{N_c}{t_{last} - t_{first}} \quad (11)$$

and the end-to-end latency of a cross-chain match is decomposed into four stages, which instantiate the generic model of Eq. (7),

$$L_{E2E} \approx L_{EVM} + L_{Fab} + L_{Cac} + L_{DB} \quad (12)$$

where, the start and end points of each stage are as follows. L_{EVM} runs from the gateway's call of the contract method in ethers.js until the JSON-RPC endpoint returns the transaction hash, that is, until the transaction is accepted into the provider's transaction pool; it does not include block inclusion. L_{Fab} runs from submission of the transaction proposal to the endorsing peers until the Fabric Gateway SDK returns the result of the submitted transaction. L_{Cac} runs from receipt of the RequestCreated event by the gateway until the signed relay is handed to the Ethereum connector, and covers the donor query, scoring, hashing, signing, and relay construction. L_{DB} is the duration of the PostgreSQL insert transaction. The end-to-end latency L_{E2E} is measured as wall-clock time from receipt of the RequestCreated event to completion of the index write, so it is the off-chain processing latency of a match and excludes public block inclusion, which is reported separately in Section VII-C. For throughput, t_{first} and t_{last} are the completion times of the first and last transaction at the respective layer: the commit event for Fabric, the JSON-RPC response for Ethereum, the relay hand-off for Cactus, and the insert commit for PostgreSQL. The integrity rate IR is the percentage of confirmed matches for which the digests stored on both ledgers are equal (N_{eq}), while the tamper-rejection rate TRR is the percentage of N_{tam} relays that are manipulated or forged that the contract rejects (N_{rej}):

$$IR = \frac{N_{eq}}{N_m} \times 100\% \quad (13)$$

$$TRR = \frac{N_{rej}}{N_{tam}} \times 100\% \quad (14)$$

Two runs were conducted: one type of run was the baseline run using a normal portal, and another type was the 100-transaction stress run using the 100 transactions issued via the benchmark module with the configuration listed in Table VI. The baseline values (Table IX, “base”) and the stress values (“stress”) are reported individually in the text, and each number quoted is identified by a reference to the corresponding run. In line with the process of blockchain benchmarking [27], [28], each ledger was checked at the interface, while the gateway was checked separately.

B. Functional Validation

Table VIII lists the twelve functional test cases run on the prototype, including onboarding, identity integrity, donor registration, pending/matched requests, multi-hospital ranking, transport tracking, audit and public verification. All twelve passed. These tests are tests of behaviour and not of statistics; with two matched requests, no success rate of any statistical nature can be arrived at from these tests. Another observation of the delay a transplant coordinator has is that the REQ040 is recorded at 14:13:03 and its match is provisionally confirmed in the portal at 14:13:08, which is a delay of 5 s. This is different from the 251.9 ms as reported below, and it does not include the inclusion of the request transaction on the public chain, but rather the time before the inclusion of the confirmation transaction in a block.

TABLE VIII. FUNCTIONAL VALIDATION TEST CASES

ID	Scenario	Expected behaviour	Observed result
TC1	Hospital registration	HospitalRegistered event on Sepolia	Pass (block 11388432)
TC2	Hospital login	Authenticated session, dashboard access	Pass
TC3	Second hospital onboarding	New consortium member accepted	Pass (HOSP007)
TC4	Duplicate hospital ID	Registration rejected	Pass ("Hospital ID already registered")
TC5	Donor registration on Fabric	Anonymized donor ID issued	Pass (DON-0323BB)
TC6	Request without compatible donor	Request logged and kept pending	Pass
TC7	Cross-chain match	Verified match via Cactus gateway	Pass (REQ040 → DON-DD7115, 100 %)
TC8	Multi-hospital ranking	Ranked candidates from another hospital	Pass (REQ032 84 %, REQ036 87 %)
TC9	Transport lifecycle	Four signed stage updates	Pass
TC10	Cross-chain audit trail	Linked Fabric, Ethereum and Cactus records	Pass (3 linked entries)
TC11	Dashboard analytics	Requests, matches and success rate updated	Pass (counters updated: 2 requests, 2 matches)
TC12	Public verification	Transaction and event decoded on Etherscan	Pass (16 confirmations)

C. Throughput and Latency

Table IX and Fig. 7 summarize the measurements. The private Fabric channel processed 153.9 TPS in the baseline run and 154.3 TPS in the stress run, the Cactus bridge 57.8 and 58.8 TPS, and the public contract interface 20.8 and 20.5 TPS. These are observed rates of a lightly loaded single-host system, not capacities: the stress run comprised only 100 transactions from one gateway account, and no Fabric container exceeded 1.2 % CPU (Table XII), so no layer was driven to saturation. Within this setting, every confirmed match needs one public transaction, so the observed end-to-end rate is bounded by the public interface at about 20 matches per second (stress run), and a batch of 100 matches cleared the public interface in about 4.9 s, the bridge in 1.7 s, and the Fabric channel in 0.65 s. Whether these rates hold under sustained load has not been tested (Section IX).

TABLE IX. THROUGHPUT AND LATENCY PER LAYER (BASELINE AND 100-TRANSACTION STRESS RUN)

Layer	TPS (base)	TPS (stress)	Latency base (ms)	Latency stress (ms)
Public EVM (Ethereum RPC)	20.8	20.5	45.58	47.27
Private Fabric (mychannel)	153.9	154.3	125.38	125.59
Cactus bridge (cross-chain relay)	57.8	58.8	80.69	80.43
PostgreSQL index	1200	1200	4.2	4.2
End-to-end cross-chain match	–	–	251.9	254.15

^a. End-to-end values are the measured wall-clock means; the per-stage means are averaged independently, so their sum (255.9 ms and 257.5 ms) differs from the end-to-end mean by less than 1.6 %.

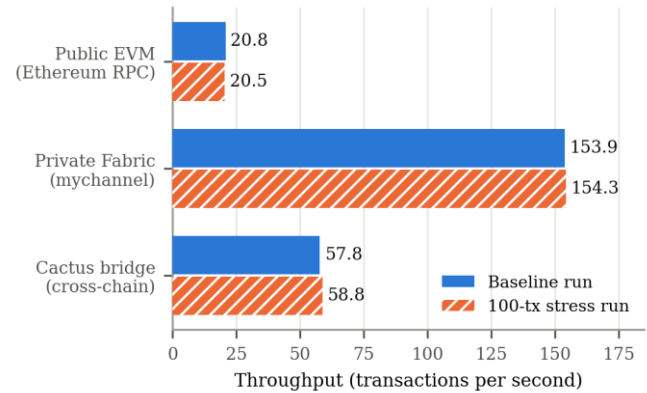


Fig. 7. Throughput of the three ledger-facing layers in the baseline and the 100-transaction stress run.

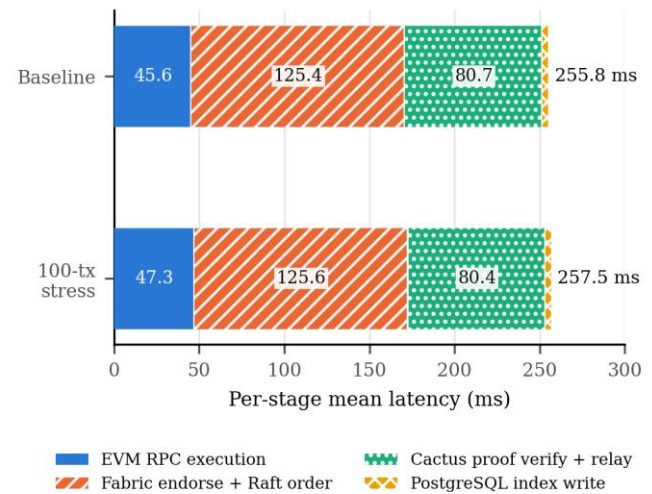


Fig. 8. Per-stage mean latency of a cross-chain donor match. Fabric endorsement and ordering dominate; the Cactus relay, including signature verification, accounts for about one third of the total.

Fig. 8 decomposes the latency. Fabric endorsement and Raft ordering are the largest contribution (125.38 ms, 49 % of the stage sum), followed by Cactus proof verification and relay (80.69 ms, 32 %), the EVM call (45.58 ms, 18 %), and the index write (4.2 ms, under 2 %). Under stress, the EVM stage grew by 1.69 ms (3.7 %) and the total by 2.25 ms (0.9 %), while the relay stage was unchanged within measurement noise. The off-chain processing of a complete match therefore stays near a quarter of a second, and the cryptographic work that protects the relay costs well under 100 ms.

1) *Public-chain inclusion and finality*: The latencies above end when the JSON-RPC endpoint accepts the confirmation transaction, so they do not include the time until the match is recorded on the public ledger. That time has two further components. Inclusion is the interval from submission of confirmMatch to the block that contains it; Sepolia produces a block every 12 s slot, so inclusion takes at least the remainder of the current slot and typically 12–14 s. Finality follows

Ethereum's Gasper protocol [32], under which a block is finalized after two epochs of 32 slots, about 12.8 min. A stage budget of the user-perceived path, from a hospital's request to the inclusion of the match on Sepolia, is therefore approximately 0.15 s for API dispatch, 1.5 s for detection of the request event by the gateway's block-header polling, 0.25 s for the Fabric matching transactions and 12–14 s for inclusion, about 14.4 s in total. This budget is an estimate assembled from the stage measurements and the Sepolia slot time, not a per-transaction measurement; it shows that public-chain inclusion accounts for more than 85 % of the user-perceived delay. LifeChain treats a match as provisionally confirmed at inclusion and as final at finalization; the 5 s interval of Section VII-B ended before the confirmation itself was included. These public-chain times, not the 251.9 ms of off-chain processing, are the quantities comparable with the inter-blockchain transfer times of other systems (Section VIII).

D. Load Stability and Concurrency

TABLE X. LOAD STABILITY AND ESTIMATED CONCURRENCY PER LAYER

Layer	ΔTPS (%)	$\Delta\text{Latency}$ (%)	$N_{\text{n base}}$	$N_{\text{n stress}}$
Public EVM (Ethereum RPC)	−1.44	+3.71	0.95	0.97
Private Fabric (mychannel)	+0.26	+0.17	19.30	19.38
Cactus bridge	+1.73	−0.32	4.66	4.73
PostgreSQL index	0.00	0.00	5.04	5.04
End-to-end match	—	+0.89	—	—

^b. Δ values are relative changes from the baseline to the 100-transaction stress run; N_{n} is the Little's law estimate of transactions in flight.

To assess stability, the relative change between the baseline and the stress run was computed for every layer, and the mean number of transactions in flight was estimated with Little's law,

$$N_{\text{n}} = \lambda W \quad (15)$$

where, N_{n} is the mean number of transactions in flight, λ is the measured throughput, and W is the mean latency of the layer. Table X reports both quantities.

All throughput and latency variations are within ± 4 %. This only indicates that 100 transactions did not have any measurable impact on the behavior; it does not imply anything about the stability near saturation. The throughput ranking is due to the concurrency estimates. The Fabric channel maintains approximately 19 transactions in flight since the endorsement and ordering are pipelined, while the public interface maintains less than one in flight: contract calls from the single gateway account are serialized with the account nonce. It is not a property of the contract rather a property of the submission strategy – can be overcome by pipelining nonces or through multiple accounts of relayers. The bridge is already overlapping with consecutive matches (around 4.7 relays in flight).

E. Bottleneck Analysis and Throughput Projection

The pipeline will have a sustainable end-to-end match rate of no more than the slowest element in the pipeline, as a

confirmed match must travel through each element once. If k relayers account for their submission in parallel, and the per-account public rate is used above,

$$\lambda_{\text{E2E}}(k) = \min(k\lambda_{\text{EVM}}, \lambda_{\text{Cac}}, \lambda_{\text{Fab}}) \quad (16)$$

Table XI and Fig. 9 evaluate Eq. (16) with the stress-run values $\lambda_{\text{EVM}} = 20.5$, $\lambda_{\text{Cac}} = 58.8$ and $\lambda_{\text{Fab}} = 154.3$ TPS. Eq. (16) is an analytical upper bound and is not a measurement: it assumes that the rates observed under light load in both public-side scalability and the bridge are not modified, and that the number of accounts in the public-side scalability is linearly scaled while the rates to Fabric remain the same. Neither of these assumptions was tested, and in practice the bound would be reduced by RPC rate limits, transaction-pool behaviour, and contention within the gateway. Under these assumptions, the pipeline is public-side bound at 20.5 matches per second with one account and 41.0 with two, and from three accounts onward the bridge becomes the binding stage at 58.8 matches per second. The projection indicates where the bottleneck would move as relayer accounts are added; it does not predict the rate that would be achieved, which requires the sustained-load test planned in Section IX.

TABLE XI. PROJECTED END-TO-END MATCH THROUGHPUT VERSUS RELAYER ACCOUNTS

Relayer accounts k	$k \cdot \lambda_{\text{EVM}}$ (TPS)	Projected λ_{E2E} (TPS)	Bottleneck
1 (measured)	20.5	20.5	Public EVM
2	41.0	41.0	Public EVM
3	61.5	58.8	Cactus bridge
4	82.0	58.8	Cactus bridge
8	164.0	58.8	Cactus bridge

^c. Analytical bound from Eq. (16) using stress-run throughputs observed under light load; assumes, without testing, linear scaling of the public side with nonce-independent accounts and unchanged bridge and Fabric rates.

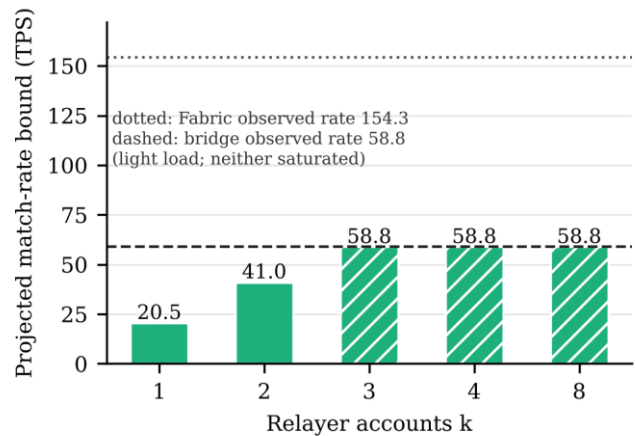


Fig. 9. Projected end-to-end match throughput of the proposed pipeline as a function of the number of relayer accounts; the dashed lines mark the observed (not saturated) bridge and Fabric rates.

F. Resource Utilization

TABLE XII. RESOURCE UTILIZATION OF THE FABRIC CONTAINERS DURING THE EVALUATION

Container	Role	CPU (%)	Memory (MB)	Net I/O (in / out)
peer0.org1.example.com	Fabric peer Org1	1.2	142	1.2 MB / 890 KB
peer0.org2.example.com	Fabric peer Org2	1.1	138	1.1 MB / 850 KB
orderer.example.com	Raft orderer	0.8	95	2.4 MB / 2.1 MB
ca_org1	Org1 CA	0.1	42	450 KB / 320 KB
ca_org2	Org2 CA	0.1	40	430 KB / 310 KB
Total	5 containers	3.3	457	5.6 MB / 4.5 MB

Table XII lists the container statistics. No container exceeded 1.2 % CPU, and the five Fabric containers together used 457 MB of memory, about 5.6 % of the 8 GB available. The orderer carried the highest network traffic (2.4 MB in, 2.1 MB out), as expected for the component that distributes blocks to both peers. These figures confirm that the network was lightly loaded during the evaluation. They show that the footprint of an idle peer is small, about 140 MB, but they do not establish the resources a peer would need at production load.

G. Data Integrity and Security Evaluation

Table XIII and Fig. 10 report the security results. The security audit of the benchmark module exercised two attack classes against the deployed contract: relays whose payload fields were altered after signing, and relays signed with keys

other than the registered gateway key. All relays from both classes were rejected (TRR = 100 %), and in the confirmed matches, the digest stored on Fabric matched the digest verified on Ethereum (IR = 100 %). The attack set is small and defined by the author, and the state check in Proposition 2 enforced the attack set, but was not performed as an audit class. Over such a set, a 100 % rejection rate indicates that 1) checks are being implemented and 2) that the checks are accessible on the paths tested. The checks are executed within the contract, which means that it is impossible to create a valid match if the network path is compromised or if a relayer does not have skG (Proposition 6). The duplicate-registration test confirmed that the contract enforces uniqueness of an identifier across hospitals.

TABLE XIII. INTEGRITY AND SECURITY EVALUATION

Test	Mechanism	Result
Cross-ledger digest parity	Keccak-256 / SHA-256 payload hash on Ethereum and Fabric	100 % identical (all confirmed matches)
Relayer authentication	ECDSA signature, on-chain ecrecover	100 % recovered to gateway key
Mid-flight payload alteration	Digest recomputation in contract	100 % rejected
Forged relay signature	Signer-address check	100 % rejected
Duplicate hospital registration	Unique-key constraint in contract	Rejected ("Hospital ID already registered")
Public verifiability	Sepolia transaction and event logs	Verified on Etherscan (16 confirmations)

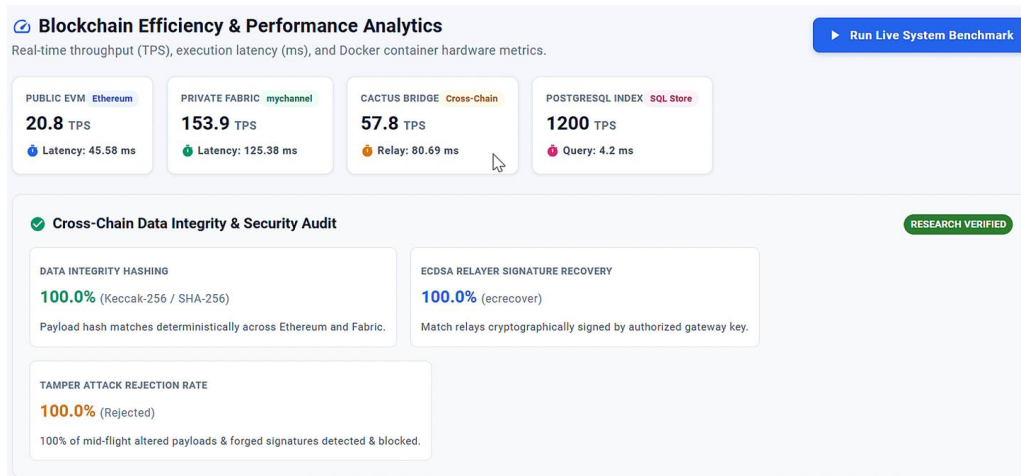


Fig. 10. Performance and security dashboard of the prototype (baseline run): layer throughput and latency, together with the cross-chain data integrity, ECDSA signature recovery and tamper-rejection results.

H. Public Verifiability and Cost

The hospital registration transaction 0xccc4ddd41...a475de2957 was included in Sepolia block 11388432 with status *Success* and 16 block confirmations at the time of inspection. Fig. 11 shows its decoded event log: topic 0 is the event signature hash, topic 1 is the hospital identifier (2), topic 2 is the administrator address, and the data field is the

hospital name. The transaction fee F was 1.888×10^{-4} ETH at a gas price p of 1.1187 Gwei (1.1187×10^{-9} ETH), so the gas consumed g was

$$g = \frac{F}{p} = \frac{1.8884 \times 10^{-4}}{1.1187 \times 10^{-9}} \approx 168\,805 \quad (17)$$

units of gas. A registration is a one-time operation per hospital, so this cost is negligible. Requests and match

confirmations store only identifiers, scores, and a 32-byte digest, and their cost is of the same order. Placing the bulk of the data on Fabric keeps the public footprint, and therefore the fee exposure, small. Table XIV projects the fee of the same

168,805-gas transaction at higher gas prices: even at 30 Gwei the registration costs about 5.1×10^{-3} ETH, and a year of operation with one registration per hospital adds a negligible, predictable cost.

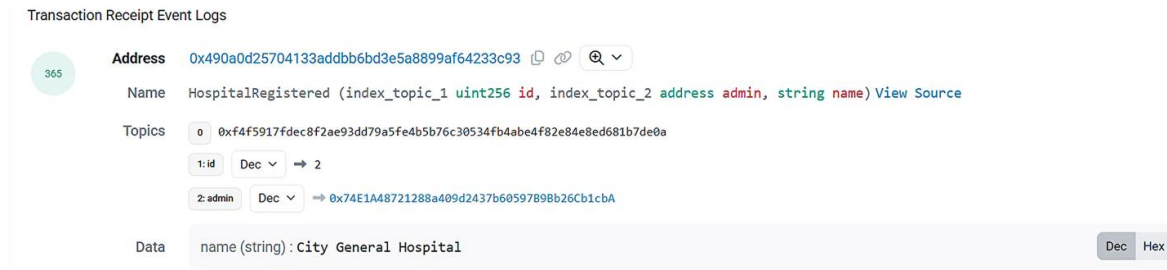


Fig. 11. Decoded Sepolia event log of the HospitalRegistered event emitted by the LifeChain contract, showing the indexed hospital identifier, the administrator address, and the hospital name.

TABLE XIV. FEE SENSITIVITY OF THE HOSPITAL REGISTRATION TRANSACTION (168,805 GAS)

Gas price (Gwei)	Fee (ETH)	Relative to measured
1.1187 (measured)	1.888×10^{-4}	1.0×
5	8.440×10^{-4}	4.5×
10	1.688×10^{-3}	8.9×
30	5.064×10^{-3}	26.8×

^d. Fee = gas used × gas price; rows other than the first are projections for congested network conditions.

Echtermeyer [12] and HIREX [13] address heterogeneous ledgers, in supply-chain and hiring contexts respectively, where the exchanged objects (shipment events and credentials) have different confidentiality requirements from clinical donor data. The proposed LifeChain is the only one of the four designs that combines private–public heterogeneity, a strict personal-data boundary, and an on-chain cryptographic check of every relayed payload.

VIII. COMPARATIVE ANALYSIS WITH EXISTING WORKS

A. Qualitative Comparison

Table XV contrasts the proposed LifeChain framework with the three existing works. The Fabric federation of Hashim *et al.* [11] is the closest in domain and tooling, since it also uses Hyperledger Cactus for healthcare data, but it connects two ledgers of the same kind and provides no public audit trail.

TABLE XV. QUALITATIVE COMPARISON OF THE PROPOSED LIFECHAIN WITH EXISTING INTEROPERABILITY WORKS

Criterion	Hashim et al. [11]	Echtermeyer [12]	HIREX [13]	Proposed LifeChain
Domain	Electronic health records	Supply chain	Hiring / credentials	Organ donation and transplant
Ledgers	Two Hyperledger Fabric v2.x consortia	Heterogeneous blockchains	Ethereum, Hyperledger Fabric, Quorum	Hyperledger Fabric + Ethereum (Sepolia)
Heterogeneity	Homogeneous (Fabric–Fabric)	Heterogeneous	Heterogeneous	Heterogeneous, private–public
Interoperability mechanism	Cactus; local/global smart contracts; hierarchical PKI	Data exchange between heterogeneous supply-chain ledgers	Multi-chain service layer (Node.js SDK) over issuer chains	Cactus gateway with signed, on-chain-verified relays
Relay integrity proof	PKI-signed transactions	NS	Certificate provenance on issuer chain	Keccak-256 digest + ECDSA, verified by ecrecover; attests the gateway, not Fabric state (Prop. 6)
On-chain relay verification	No	NS	NS	Yes (ecrecover in contract)
Public audit trail	No	NS	NS	Yes (Sepolia, Etherscan)
Privacy boundary	Within federations	NS	Per issuer chain	PII only on Fabric; anonymized IDs public
Layer-wise latency analysis	No (aggregate)	NS	NS	Yes (four stages)
Evaluation	TPS, latency, CPU, inter-chain time	No comparable metrics	Functional evaluation	TPS, stage latency, resources, gas, integrity, tamper, 12 test cases

^e. NS: the mechanism is not specified in the published work; we do not infer it. Entries for the existing works are taken from the cited publications.

B. Quantitative Comparison

Table XVI and Fig. 12 compare the values; the markers in Table XVI distinguish values reported in the cited works (R) from values measured in this work (M), values computed from them (C) and protocol properties (P). The cross-chain latency values of the two systems are not equivalent. The 1.0125 s of Hashim *et al.* is the steady-state time of a Fabric-to-Fabric transfer and includes commitment on the destination Fabric network, which is final once committed. The 251.9 ms of LifeChain (baseline run) is the off-chain processing time of a match and ends when the public JSON-RPC endpoint accepts the confirmation, before the transaction is included in a block. The like-for-like quantity for LifeChain is the time until the

confirmation is included on Sepolia, which is bounded below by the 12 s slot time and typically 12–14 s; the estimated user-perceived path from request to inclusion is about 14.4 s (Section VII-C), and finality follows after about 12.8 min. On this basis, LifeChain is slower than the Fabric federation by roughly an order of magnitude at inclusion; this is the price of recording allocation events on a public ledger. What the processing figure does show is that the gateway's own contribution, including signing and the construction of an on-chain-verifiable relay, is about a quarter of a second, so the delay is dominated by public-chain inclusion and not by the interoperability layer. A permissioned EVM network or a layer-2 rollup with faster finality would narrow the gap (Section X).

TABLE XVI. QUANTITATIVE COMPARISON OF THE PROPOSED LIFECHAIN WITH EXISTING INTEROPERABILITY WORKS

Metric	Hashim et al. [11]	Echtermeyer [12]	HIREX [13]	Proposed LifeChain
Intra-chain throughput (TPS)	496.3 / 494.6 (R; Caliper, 500 TPS send rate)	NR	NR	153.9 (Fabric) / 20.8 (Ethereum RPC acceptance) (M, baseline)
Intra-chain latency	0.868 s / 0.922 s (R)	NR	NR	125.38 ms (Fabric) / 45.58 ms (EVM RPC acceptance) (M, baseline)
Cross-chain latency	1.0125 s steady-state transfer, includes commit on the destination ledger (R)	NR	NR	251.9 ms off-chain processing, excludes public inclusion (M, baseline); 5 s request to provisional confirmation (M, one observation); ≈ 12 –14 s inclusion, ≈ 14.4 s user-perceived path (estimate); ≈ 12.8 min finality (P)
Cross-chain latency relative to [11]	1 (reference)	NR	NR	0.25 for off-chain processing, not like-for-like (C); ≥ 11.9 for inclusion, lower bound from one 12 s slot (C); ≈ 14 for the estimated 14.4 s user-perceived path (C)
Cross-chain throughput	NR	NR	NR	57.8 / 58.8 TPS bridge, baseline/stress (M); observed, not a capacity
CPU utilization	68–79 % at 500 TPS (R)	NR	NR	≤ 1.2 % per container (M, light load)
Integrity/tamper rejection	NR	NR	NR	100 % / 100% overall confirmed matches and audited attacks (M)
Load stability	Latency grows with load; 339 failed tx at 5000 TPS on B2 (R)	NR	NR	< 4 % change from baseline to stress (C, from Table X); light load only
Testbed	2 $\times$ Intel Core i5-4570, 8 GB RAM	NR	NR	Single workstation, Docker (WSL 2), 8 GB

^f R: value reported in the cited publication. M: value measured in this work. C: value computed by the authors of this study from reported or measured values. P: property of the Sepolia protocol, not measured. NR: not reported in a form comparable to the metrics of the proposed framework.

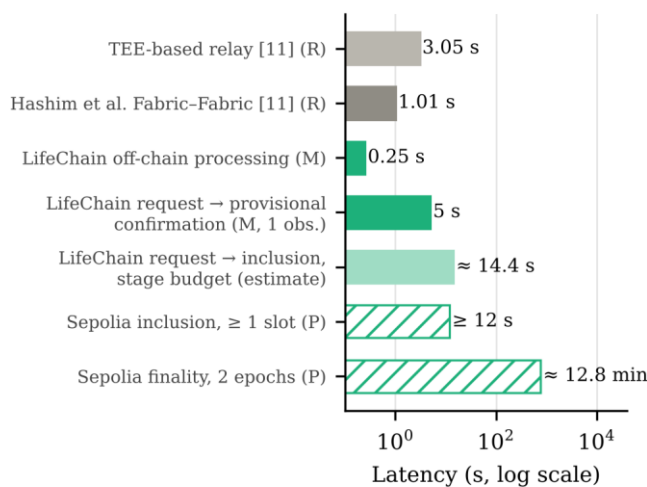


Fig. 12. Cross-chain latency tiers of the proposed LifeChain compared with the Fabric–Fabric federation of Hashim *et al.* [11] and the trusted-execution relay they report (logarithmic scale). Hatched bars are protocol bounds of

Sepolia, and the light bar is an estimated stage budget; neither is a per-transaction measurement. Only the inclusion and finality tiers of LifeChain are comparable with the transfer times of [11].

Intra-chain throughput cannot be compared on equal terms either. Hashim *et al.* obtained about 496 TPS per Fabric network with Hyperledger Caliper at a 500 TPS send rate on two dedicated machines (R), whereas the Fabric channel of LifeChain processed 153.9 TPS in the baseline run and 154.3 TPS in the 100-transaction stress run on a single host under WSL, with default block-cutting parameters and without a sustained load generator (M). The two values describe different operating points: the first is near saturation, with 68–79 % CPU use, and the second is far from it, with at most 1.2 % CPU per container. Neither the throughput of LifeChain nor the bound of Table XI should therefore be read as a capacity; a Caliper-based saturation test on comparable hardware is required to compare capacities (Section IX).

Echtermeyer [12] and HIREX [13] do not provide comparable end-to-end cross-chain timing in the published

material, so the comparison with these two works is limited to the design criteria in Table XV. With respect to them, the proposed LifeChain contributes a measured, layer-wise performance profile and a verifiable integrity mechanism in addition to heterogeneous connectivity.

IX. DISCUSSION AND LIMITATIONS

Within the limits of the evaluation, the prototype meets its design goals: DG1 and DG2 through the data placement in Table III, DG3 through Propositions 1 and 2 and the rejection of every relay in the attack set, DG4 through the uniqueness check (TC4), and DG5 through the 251.9 ms off-chain processing latency. Several limitations qualify these results, and some of them bound what the study can claim.

1) *Latency and finality*: In the public chain, the processing latency of the head does not include block inclusion and chain finality. As for the record, a match is considered provisionally confirmed at inclusion and provisionally confirmed at finalization, and clinical coordination may start at the provisional stage as long as the allocating authority agrees that a reorganization of the public chain before the match is final can simply delay, but not affect, the record, as the match is also anchored on Fabric.

2) *Scale of the evaluation*: All the components were running on one Windows workstation on WSL, the stress run consisted of 100 transactions from one gateway account, and the CPU consumption remained under 1.2 % per Fabric container. The reported throughputs are thus the actual throughputs obtained with a lightly loaded system, and the extrapolations of Table XI are based on an untested linear-scaling assumption. Before making any capacity claim, a multi-host deployment which is benchmarked with Hyperledger Caliper and sustained, increasing load, with several relayer accounts and repeated runs, is required.

3) *Trust in the gateway*: The contract is not a check on the fabric state; it's a check on the gateway's signature. If the gateway key is corrupted, then false matches on the public chain are possible, but if the chain is anchored on Fabric, then this is detectable, but not preventable to date (Proposition 6), meaning that the cross-chain integrity is only protected by one operator. There are two remedies for this single point of trust: a t-of-n threshold signature across gateways from different consortium members to prevent any single member from confirming a match; and verification of Fabric endorsement signatures (which are in fact ECDSA signatures over the NIST P-256 curve) within the contract, to prevent the contract from relying on the endorsed Fabric state. The second option makes sense on networks like EVM that have a P-256 verification precompile.

4) *Strength of the security analysis*: The propositions of Section V are informal proof sketches using an adversary model; the tamper tests are performed on a small, author-chosen attack set of two classes, one performed at a time. The following measurement values in Section VII are only based on one set of measurements, and no repetition or confidence interval is applied. These results would be enhanced with a symbolic

analysis of the protocol using, e.g., Tamarin or ProVerif, and a systematic attack set derived from the message fields. Only partially mitigated: Metadata linkage (Proposition 4).

5) *Matching model and clinical validity*: Functional validation contains a set of twelve tests with two matched requests, and it showcases behaviour, not allocation quality. The score of Eq. (9) is based on weights selected by the authors and does not include human leukocyte antigen (HLA) typing, cross-match results, geography, and waiting time, and was not compared against any existing allocation rule, nor was it reviewed by clinicians or regulators. The intention of LifeChain is not to be an allocation system, but an interoperability protocol that can be accompanied by various scoring functions. Since protocol signs S as an opaque value, a validated rule, e.g., a points-based national allocation policy [29] or the criteria of the competent Indian authority under [1], may be substituted, and the results of the allocation of the two rules compared case by case to the allocation decisions of the past.

6) *Legal constraints on public allocation events*: The public chain publication of allocation events will certainly make lawyers consider legal issues that this study does not address. According to GDPR, pseudonymized data can still be personal data if re-identification is reasonably possible and the immutability of a public ledger is incompatible with the rights to rectification and erasure [8]. The European Data Protection Board advises against storing personal data on a public blockchain where this can be avoided and notes that hashes or commitments of personal data may themselves be personal data [31]. In India, a deployment would fall under the Digital Personal Data Protection Act, 2023 [30] and the Transplantation of Human Organs and Tissues Act [1], under which allocation is governed by the national and state transplant authorities. A compliant design would publish only salted commitments and coarse-grained events, and would keep the salts on Fabric so that deleting a salt renders the public commitment unlinkable; any deployment would also require review by the competent transplant authority and a data protection officer, which has not been obtained for this prototype.

X. CONCLUSION AND FUTURE WORK

This study proposed LifeChain, a hybrid cross-chain framework for organ donation and transplant matching that keeps donor data on a permissioned Hyperledger Fabric ledger, publishes allocation events on a public Ethereum network, and connects them through a Hyperledger Cactus gateway whose relays are bound to the receiving contract by a Keccak-256 digest and an on-chain-verified ECDSA signature. Under the stated adversary model, forging, altering, or replaying a relay requires breaking ECDSA or Keccak-256, at an on-chain verification cost of about 4,100 gas; the gateway remains a trusted attester whose compromise is detectable through Fabric anchoring but not prevented. On a single-host prototype with the Sepolia testnet and under light load, the private ledger processed 153.9 TPS (baseline) and 154.3 TPS (stress), the bridge 57.8 and 58.8 TPS and the public interface 20.8 and 20.5 TPS; the off-chain processing of a cross-chain match took 251.9 ms (baseline), public inclusion adds at least one 12 s slot, giving an

estimated user-perceived path of about 14.4 s, and finality follows after about 12.8 min. All relays of the audited attack classes were rejected and digest parity held for every confirmed match, and all twelve functional test cases passed. Compared with a homogeneous Fabric federation for health records, LifeChain adds public verifiability across heterogeneous ledgers at the cost of a longer, public-chain-dominated confirmation time.

Future work will benchmark the framework with Hyperledger Caliper on a multi-host deployment at sustained load, replace the single gateway key with threshold signatures or on-chain verification of Fabric endorsements, analyse the protocol formally, replace the placeholder score with a validated allocation rule developed with clinicians and the competent transplant authority, assess compliance with data protection law, and study deployment on a permissioned EVM network or a layer-2 rollout to obtain faster finality at predictable cost.

DECLARATIONS

Conflict of interest: The authors declare no conflict of interest. Data availability: The LifeChain contract is publicly verifiable on the Ethereum Sepolia testnet at address 0x490A0d25704133adDBB6BD3E5a8899af64233C93; source code and benchmark logs are available from the corresponding author on reasonable request.

REFERENCES

- [1] Government of India, The Transplantation of Human Organs and Tissues Act, Act No. 42 of 1994 (as amended 2011).
- [2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [3] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *J. Amer. Med. Inform. Assoc.*, vol. 24, no. 6, pp. 1211–1220, 2017.
- [4] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, 2016, pp. 25–30.
- [5] G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella, "Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology," *Sustain. Cities Soc.*, vol. 39, pp. 283–297, 2018.
- [6] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, "FHIRChain: Applying blockchain to securely and scalably share clinical data," *Comput. Struct. Biotechnol. J.*, vol. 16, pp. 267–278, 2018.
- [7] D. Hawashin, R. Jayaraman, K. Salah, I. Yaqoob, M. C. E. Simsekler, and S. Ellahham, "Blockchain-based management for organ donation and transplantation," *IEEE Access*, vol. 10, pp. 59013–59025, 2022.
- [8] European Parliament and Council, Regulation (EU) 2016/679 (General Data Protection Regulation), *Off. J. Eur. Union*, L 119, 2016.
- [9] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 168:1–168:41, 2021.
- [10] V. Buterin, "Chain interoperability," R3 Research Paper, 2016.
- [11] F. Hashim, K. Shuaib, and F. Sallabi, "Connected blockchain federations for sharing electronic health records," *Cryptography*, vol. 6, no. 3, p. 47, 2022, doi: 10.3390/cryptography6030047.
- [12] H. Echtermeyer, "Interoperability between heterogeneous blockchains for supply chain," M.S. thesis, Norwegian Univ. Sci. Technol. (NTNU), Trondheim, Norway, 2022.
- [13] R. Amin, A. Islam, D. H. Tanvir, and R. I. Arif, "HIREX: A heterogeneous interoperable blockchain solution for hiring system," in *Proc. Int. Conf. Comput. Commun. Technol. (ICCT)*, Dhaka, Bangladesh, 2023, pp. 1–6.
- [14] E. Androulaki et al., "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Porto, Portugal, 2018, pp. 30:1–30:15.
- [15] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.
- [16] H. Montgomery et al., "Hyperledger Cactus white paper," Hyperledger Foundation, 2020. [Online]. Available: <https://github.com/hyperledger/cacti>
- [17] J. Kwon and E. Buchman, "Cosmos: A network of distributed ledgers," White paper, 2016.
- [18] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," White paper, 2016.
- [19] M. Herlihy, "Atomic cross-chain swaps," in *Proc. ACM Symp. Principles Distrib. Comput. (PODC)*, 2018, pp. 245–254.
- [20] E. Abebe et al., "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)," in *Proc. 20th Int. Middleware Conf. Industrial Track*, 2019, pp. 29–35.
- [21] B. Pillai, K. Biswas, and V. Muthukumarasamy, "Cross-chain interoperability among blockchain-based systems using transactions," *Knowl. Eng. Rev.*, vol. 35, p. e23, 2020.
- [22] J. R. Douceur, "The Sybil attack," in *Proc. 1st Int. Workshop Peer-to-Peer Syst. (IPTPS)*, 2002, pp. 251–260.
- [23] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2014, pp. 305–319.
- [24] National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4, 2015.
- [25] National Institute of Standards and Technology, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, FIPS PUB 202, 2015.
- [26] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, 2001.
- [27] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A framework for analyzing private blockchains," in *Proc. ACM SIGMOD*, 2017, pp. 1085–1100.
- [28] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing Hyperledger Fabric blockchain platform," in *Proc. IEEE MASCOTS*, 2018, pp. 264–276.
- [29] A. K. Israni et al., "New national allocation policy for deceased donor kidneys in the United States and possible effect on patient outcomes," *J. Amer. Soc. Nephrol.*, vol. 25, no. 8, pp. 1842–1848, 2014, doi: 10.1681/ASN.2013070784.
- [30] Government of India, The Digital Personal Data Protection Act, 2023, Act No. 22 of 2023.
- [31] European Data Protection Board, Guidelines 02/2025 on Processing of Personal Data Through Blockchain Technologies, version 1.0 (for public consultation), Apr. 2025.
- [32] V. Buterin et al., "Combining GHOST and Casper," *arXiv:2003.03052*, 2020.