JusticeChain -- Layer 1 Blockchain Whitepaper

Generated March 30, 2026 | law.unykorn.org

JusticeChain -- Layer 1 Blockchain Whitepaper

Immutable Accountability Infrastructure for Criminal Justice

---

1. ABSTRACT

JusticeChain is a purpose-built Layer 1 blockchain designed to provide immutable records for criminal justice defense operations. Every case registration, evidence hash, disclosure acknowledgment, timeline event, attorney referral, and public records request is permanently recorded on-chain -- creating an accountability layer that cannot be altered, deleted, or disputed.

This is not a cryptocurrency. This is not a DeFi platform. This is accountability infrastructure.

---

2. PROBLEM STATEMENT

In criminal defense -- particularly wrongful conviction and post-conviction cases -- the integrity of records is everything:

* **Evidence disappears.** Video footage is "lost." Records are "unavailable."
* **Timelines are disputed.** "We sent that disclosure." "The request was never received."
* **Disclosures are ignored.** Non-attorney disclaimers are required but hard to prove were presented.
* **Chain of custody breaks.** Who had the evidence? When? What was done to it?

JusticeChain solves this by providing a tamper-proof, publicly verifiable record of every significant event in a case's lifecycle.

---

3. ARCHITECTURE

3A. Technical Foundation

JusticeChain is a fork of the Apostle Chain codebase (Rust/Axum), adapted for justice-specific operations.

Apostle Chain (chain_id 7332)	JusticeChain (chain_id 7333)
??? Rust/Axum server	??? Rust/Axum server (forked)
??? 10 crates	??? 12 crates (2 new justice-specific)
??? Agent registration	??? Case registration
??? ATP/UNY/USDF tokens	??? JST (JusticeToken) -- utility only
??? XRPL + Stellar bridges	??? JusticeFund bridge (donations)
??? Port 7332	??? Port 7333

3B. Chain Specifications

Parameter	Value

Chain Name	JusticeChain
Chain ID	7333
Consensus	Proof of Authority (PoA)
Block Time	~2 seconds
Native Token	JST (JusticeToken)
Token Decimals	18
Token Supply	1,000,000,000 JST (fixed, no inflation)
Token Purpose	Transaction fees only (not traded)
Validators	3-5 platform-operated nodes (initial)
Explorer	justice-explorer.xxxiii.io
RPC Endpoint	rpc.justicechain.xxxiii.io
Port	7333

3C. Why PoA (Not PoW or PoS)

* **PoA** = platform operates all validators
* No mining, no staking, no token speculation
* Fast finality (2-second blocks)
* Low overhead (runs on standard EC2 instances)
* Appropriate for accountability infrastructure where decentralization of consensus is less important than immutability of records

3D. Future Decentralization Path

As the platform grows, validator set can expand to include:
* Verified advocacy organizations
* Law school clinics
* Legal aid organizations
* Public defender offices

This transitions from centralized PoA to a federated validator model -- trusted organizations collectively maintain the chain.

---

4. ON-CHAIN DATA MODEL

4A. Transaction Types

```
/// All transactions on JusticeChain
pub enum JusticeTxType {
/// Register a new case on-chain
CaseRegistration {
case_hash: [u8; 32],    // SHA-256 of case intake data
jurisdiction: String,
charge_statute: String,
created_at: u64,
},

/// Record evidence hash
EvidenceRecord {
case_hash: [u8; 32],
```

evidence_hash: [u8; 32], // SHA-256 of evidence file
ipfs_cid: String,
evidence_type: String,
recorded_at: u64,
},

/// Record disclosure acknowledgment
DisclosureAck {
case_hash: [u8; 32],
user_hash: [u8; 32],   // SHA-256(user_id) -- privacy
disclosure_type: String,
acknowledged_at: u64,
},

/// Record timeline event
TimelineEvent {
case_hash: [u8; 32],
event_hash: [u8; 32],
event_type: String,
event_date: u64,
description_hash: [u8; 32],
},

/// Record attorney referral
AttorneyReferral {
case_hash: [u8; 32],
firm_hash: [u8; 32],   // SHA-256(firm_id) -- privacy
referred_at: u64,
status: String,
},

/// Record public records request
PublicRecordsRequest {
case_hash: [u8; 32],
agency: String,
request_hash: [u8; 32],
filed_at: u64,
status: String,
},

/// JusticeFund donation record
Donation {
fund_id: [u8; 32],
donor_hash: [u8; 32],   // SHA-256(donor_id) -- privacy
amount: u128,
donated_at: u64,
},
}

4B. Privacy Design

On-chain: Only hashes. Never raw personal data.

Data	On-Chain	Off-Chain
Case existence	SHA-256 hash of case data	Full case details in PostgreSQL
Evidence	SHA-256 hash + IPFS CID	Original files in IPFS + local storage
User identity	SHA-256(user_id)	User profile in PostgreSQL
Firm identity	SHA-256(firm_id)	Firm details in firm_database.json
Disclosure text	SHA-256(disclosure_text)	Full text in PostgreSQL
Timeline details	SHA-256(event_description)	Full description in PostgreSQL

Verification flow: Given the off-chain data, anyone can compute the SHA-256 hash and verify it matches the on-chain record. This proves the data hasn't been altered since it was recorded.

---

5. SMART CONTRACTS

5A. CaseRegistry

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract CaseRegistry {
struct CaseRecord {
bytes32 caseHash;
string jurisdiction;
string chargeStatute;
uint256 registeredAt;
address registeredBy;
}

mapping(bytes32 => CaseRecord) public cases;
bytes32[] public caseList;

event CaseRegistered(bytes32 indexed caseHash, string jurisdiction, uint256 timestamp);

function registerCase(
bytes32 caseHash,
string calldata jurisdiction,
string calldata chargeStatute
) external {
require(cases[caseHash].registeredAt == 0, "Case already registered");
cases[caseHash] = CaseRecord({
caseHash: caseHash,
jurisdiction: jurisdiction,
chargeStatute: chargeStatute,
registeredAt: block.timestamp,
registeredBy: msg.sender
});
caseList.push(caseHash);
emit CaseRegistered(caseHash, jurisdiction, block.timestamp);
}

function verifyCase(bytes32 caseHash) external view returns (bool exists, uint256 registeredAt) {
```

```
CaseRecord storage c = cases[caseHash];
return (c.registeredAt > 0, c.registeredAt);
}

function totalCases() external view returns (uint256) {
return caseList.length;
}
}
```

5B. EvidenceVault

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract EvidenceVault {
struct EvidenceRecord {
bytes32 caseHash;
bytes32 evidenceHash;
string ipfsCid;
string evidenceType;
uint256 recordedAt;
address recordedBy;
}

mapping(bytes32 => EvidenceRecord[]) public caseEvidence;
mapping(bytes32 => bool) public evidenceExists;

event EvidenceRecorded(
bytes32 indexed caseHash,
bytes32 indexed evidenceHash,
string ipfsCid,
uint256 timestamp
);

function recordEvidence(
bytes32 caseHash,
bytes32 evidenceHash,
string calldata ipfsCid,
string calldata evidenceType
) external {
require(!evidenceExists[evidenceHash], "Evidence already recorded");
caseEvidence[caseHash].push(EvidenceRecord({
caseHash: caseHash,
evidenceHash: evidenceHash,
ipfsCid: ipfsCid,
evidenceType: evidenceType,
recordedAt: block.timestamp,
recordedBy: msg.sender
}));
evidenceExists[evidenceHash] = true;
emit EvidenceRecorded(caseHash, evidenceHash, ipfsCid, block.timestamp);
}
```

```
function verifyEvidence(bytes32 evidenceHash) external view returns (bool) {
return evidenceExists[evidenceHash];
}

function getEvidenceCount(bytes32 caseHash) external view returns (uint256) {
return caseEvidence[caseHash].length;
}
}
```

5C. DisclosureRecord

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract DisclosureRecord {
struct Disclosure {
bytes32 userHash;
bytes32 caseHash;
string disclosureType;
uint256 acknowledgedAt;
}

mapping(bytes32 => Disclosure[]) public userDisclosures;
uint256 public totalDisclosures;

event DisclosureAcknowledged(
bytes32 indexed userHash,
bytes32 indexed caseHash,
string disclosureType,
uint256 timestamp
);

function acknowledgeDisclosure(
bytes32 userHash,
bytes32 caseHash,
string calldata disclosureType
) external {
userDisclosures[userHash].push(Disclosure({
userHash: userHash,
caseHash: caseHash,
disclosureType: disclosureType,
acknowledgedAt: block.timestamp
}));
totalDisclosures++;
emit DisclosureAcknowledged(userHash, caseHash, disclosureType, block.timestamp);
}

function getDisclosureCount(bytes32 userHash) external view returns (uint256) {
return userDisclosures[userHash].length;
}
}
```

5D. Existing -- JusticeFund.sol

Already deployed in `docs/legal-pack/source-files/justice-fund-v2/`. Handles:
  * Donation vault with configurable goal
  * Refund-if-unmet mechanism
  * Multisig withdrawal (2-of-3 signers)
  * Integration: Bridge donations to JusticeChain for on-chain record

6. INTEGRATION ARCHITECTURE

6A. API ? JusticeChain Flow

User Action (via API)
?
?
FastAPI Backend
?
??? Store full data -> PostgreSQL
??? Store evidence file -> IPFS (get CID)
?
?
Blockchain Record Agent
?
??? Compute SHA-256 hash of data
??? Submit transaction to JusticeChain
??? Receive transaction hash
?
?
Store tx_hash back in PostgreSQL
?
?
User sees: "[x] Recorded on-chain -- TX: 0xabc..."

6B. Apostle Chain Bridge

JusticeChain and Apostle Chain share:
  * Same Rust/Axum codebase (forked)
  * Same crate structure (with justice-specific additions)
  * Bridge for JusticeFund donation settlement
  * Bridge for agent registration (agents can operate on both chains)

Apostle Chain (7332) JusticeChain (7333)
? ?
? ?????????????????? ?
??????? Settlement ???????
? ? Bridge ? ?
? ? (bi-directional)? ?
? ?????????????????? ?
? ?
??? ATP/UNY tokens ??? JST token
??? Agent operations ??? Case operations
??? AI settlement ??? Evidence/disclosure records

7. EXPLORER -- justice-explorer.xxxiii.io

Features

* Block explorer (blocks, transactions, addresses)
* Case registry browser (search by case hash)
* Evidence verification tool (input hash -> verify on-chain)
* Disclosure verification (prove acknowledgment date)
* Statistics dashboard (total cases, evidence records, disclosures)

---

8. DEPLOYMENT ROADMAP

Phase 1 -- Testnet (Month 2-3)

* Deploy 3 validator nodes (AWS us-east-1)
* Deploy smart contracts (CaseRegistry, EvidenceVault, DisclosureRecord)
* API integration for case/evidence/disclosure recording
* Basic block explorer
* Internal testing with Marquis case data

Phase 2 -- Mainnet Beta (Month 4-6)

* Expand to 5 validator nodes (multi-region)
* Full API integration
* Explorer with verification tools
* Bridge to Apostle Chain operational
* JusticeFund on-chain donation tracking

Phase 3 -- Mainnet (Month 6-12)

* Federated validator onboarding (advocacy orgs)
* Public verification portal
* Open-source contracts and crate code
* Multi-case operation validated

---

9. SECURITY

Vector	Mitigation
Validator compromise	PoA with 3-of-5 consensus -- no single point of failure
Smart contract bugs	Formal verification where possible. Testnet validation. No user funds at risk (JST is utility-only).
Privacy leak	Only hashes on-chain. Raw data never touches the blockchain.
Data availability	PostgreSQL + IPFS + JusticeChain = triple redundancy
51% attack	PoA -- validators are known, trusted entities. Not applicable in traditional sense.

---

10. WHY THIS MATTERS

A blockchain record doesn't win cases in court. But it does this:

1. **Proves a disclosure was presented on a specific date** -- "Your Honor, the non-attorney disclosure was

acknowledged on-chain at block 14,207 on March 15, 2026. Here is the verification."

2. **Proves evidence existed before a specific date** -- "The video hash was recorded on-chain on February 28, 2026 -- before the prosecution claims it was 'lost.'"

3. **Creates an immutable timeline** -- No one can edit the timeline after the fact. Every event is permanently recorded.

4. **Provides accountability for records requests** -- "We filed this public records request on March 1, 2026. Here is the on-chain record. The agency has not responded in 30 days as required by F.S. 119.07."

This is how technology serves justice -- not by replacing lawyers, but by making records impossible to lose, alter, or deny.

Classification: INTERNAL -- Technical Whitepaper
Version: 1.0