Pandacea Protocol - System Design Document (SDD)

Version: 1.2

Status: Draft

Date: July 20, 2025

Lead Engineer: Asa Conant

Related Documents: Pandacea Technical Whitepaper (v4.0), Pandacea Go-to-Market Plan (v1.1)

1. Overview

1.1. Purpose

This System Design Document (SDD) provides the engineering blueprint for the Pandacea Protocol Minimum Viable Product (MVP). It translates the architectural concepts outlined in the Technical Whitepaper into concrete data models, service interactions, and smart contract interfaces. This document will guide the development team in building a secure, scalable, and robust initial version of the protocol.

1.2. MVP Scope & Objectives

The primary objective of the MVP is to validate the core economic loop of the Pandacea Protocol within the autonomous warehouse logistics market.

Key Objectives:

1. Enable Earners (e.g., warehouse operators) to list and monetize specific, high-value logistics data.
2. Enable Spenders (e.g., robotics AI developers) to discover, lease, and utilize this data in a privacy-preserving manner.
3. Successfully implement and test the foundational economic mechanisms: Heuristic-Based Pricing (PDVF), Dynamic Minimum Pricing (DMP), and Reputation-Based Royalties (RBR).
4. Establish a secure and auditable "Safe Harbor" through verifiable consent on

the Polygon PoS network.

5. **Ensure protocol interoperability by exposing all core functionalities through a Model Context Protocol (MCP) compliant interface.**

2. System Architecture

2.1. Logical Architecture Diagram

The diagram below illustrates the high-level components of the Pandacea Protocol. This version introduces the **MCP Interface Layer**, which serves as the primary entry point for external agent communication, ensuring standardization and interoperability.

```
graph TD
  subgraph External AI Ecosystem
    Ext[External AI Agent <br/> (MCP Client)]
  end

  subgraph User Layer
    A[MyData Agent <br/> (Earner)]
    B[Buyer-Side Agent <br/> (Spender)]
  end

  subgraph Protocol Layer
    MCP(MCP Interface Layer)
    C(P2P Network <br/>- libp2p KAD-DHT)
    D(Storage Layer <br/>- IPFS/Filecoin)
    E(Privacy Layer <br/>- OpenMined PySyft)
  end

  subgraph Settlement Layer (Polygon PoS)
    F[LeaseAgreement Contract]
    G[Royalty Distributor Contract]
```

```
        H[Reputation Contract]
        I[PGT Token Contract]
    end

    Ext -- MCP Request --> MCP
    A -- Contains --> MCP
    MCP -- Translates to Internal Logic --> A
    A -- Publishes DataProduct --> C
    B -- Discovers DataProduct --> C
    C -- P2P Communication --> A
    C -- P2P Communication --> B
    A -- Negotiates Storage Deal --> D
    B -- Initiates Lease --> F
    F -- Interacts with --> G
    F -- Interacts with --> H
    F -- Interacts with --> I
    A -- Approves Lease via --> F
    E -- Facilitates Computation --> A
    E -- Returns Result --> B
```

2.2. Component Breakdown

- **MyData Agent (Earner):** A user-controlled application that manages data sources, enforces user policies (via OPA), and acts as a P2P node. **It now includes an embedded MCP Server that exposes its capabilities as standardized `Resources` and `Tools`.**
- **Buyer-Side Agent (Spender):** A developer-focused application for defining data requirements, discovering data products, and initiating leases. **It now acts as an MCP Client to interact with Earner agents.**
- **MCP Interface Layer:** A component within the MyData Agent that implements the Model Context Protocol. It receives standardized requests from external AI agents and translates them into calls to the agent's internal logic.

- **P2P Network (libp2p):** The communication backbone for agent discovery, direct negotiation, and data exchange.
- **Storage Layer (IPFS/Filecoin):** Decentralized storage for data payloads.
- **Privacy Layer (PySyft):** The framework for executing remote data science tasks. For the MVP, this will focus on federated analysis and statistics on extracted features.
- **Settlement Layer (Polygon PoS):** The public blockchain for recording all value-based transactions.

2.3. Technology Stack (MVP)

- **P2P Networking:** Go-libp2p
- **Agent Communication: Model Context Protocol (MCP)** with the official Go SDK
- **Blockchain:** Polygon PoS
- **Smart Contracts:** Solidity
- **Storage:** IPFS, Filecoin
- **Agent Backend:** Go / Python
- **Agent Frontend:** Electron / React
- **Privacy Computation:** PySyf

3. Core Data Models & Schemas (MVP Focus)

This section defines the initial data structures for the MVP, focusing on the two prioritized data types.

3.1. DataProduct Schema

This is the public-facing listing that an Earner publishes to the DHT to advertise their available data.

```
{
  "schemaVersion": "1.1",
  "productId": "did:pandacea:earner:123/abc-456",
  "ownerDid": "did:pandacea:earner:123",
  "name": "Novel Package 3D Scans - Warehouse A",
  "description": "High-fidelity point cloud data of irregularly shaped packages
captured by a robotic arm-mounted LiDAR sensor.",
  "dataType": "RoboticSensorData",
  "keywords": ["robotics", "3d-scan", "lidar", "logistics", "grasping"],
  "sampleCid": "bafybeig...xyz" // IPFS CID of a small, representative sample
}
```

3.2. MVP Data Type 1: RoboticSensorData

The schema for the actual data payload referenced by a DataProduct of this type. This metadata is stored alongside the raw data file (e.g., a .pcd file) on IPFS.

```
{
  "schemaVersion": "1.1",
  "captureTimestamp": "2025-07-12T19:50:00Z",
  "sensorType": "LIDAR",
  "sensorModel": "Velodyne Puck VLP-16",
  "dataFormat": "PCD_BINARY",
  "payloadCid": "bafybeig...abc", // IPFS CID of the actual .pcd file
  "storageDealId": "123456", // On-chain Filecoin deal ID if payload > 10MB
  "metadata": {
    "robotId": "arm-07",
    "warehouseId": "ATL-01",
    "taskContext": "novel_package_scan",
    "taskOutcome": "scan_success",
    "dimensions_mm": { "x": 350, "y": 210, "z": 155 },
    "estimatedWeight_g": 850
  }
}
```

3.3. MVP Data Type 2: LogisticsEventData

The schema for non-robotic but robotic-adjacent data, likely sourced from a Warehouse Management System (WMS) via an API connector in the MyData Agent.

```
{
  "schemaVersion": "1.0",
  "eventTimestamp": "2025-07-12T19:52:10Z",
  "eventType": "INBOUND_SCAN",
  "locationId": "receiving-dock-03",
  "itemIdentifier": {
    "type": "UPC",
    "value": "012345678905"
  },
  "metadata": {
    "operatorId": "emp-451",
    "equipmentUsed": "handheld_scanner_zebra_mc9300",
    "discrepancyCode": null // e.g., 'DAMAGED_BOX', 'WRONG_ITEM'
  }
}
```

4. Agent Architecture & Interaction Flow

4.1. Agent Discovery & Lease Interaction Flow (MCP-Compliant)

The following sequence diagram illustrates the end-to-end flow for a data lease transaction, now initiated via the MCP interface.

```
sequenceDiagram

    participant Spender as Buyer-Side Agent (MCP Client)

    participant Earner as MyData Agent (MCP Server)

    participant Polygon as Settlement Layer



    Spender->>Earner: Discover MCP `Resources` (DataProducts)

    Earner-->>Spender: Return list of available `Resources`

    Spender->>Earner: Call MCP `Tool`: request_lease(productId, maxPrice)

    Earner->>Earner: Evaluate request against local policy (OPA)

    Note over Earner: Manual Approval: Send push notification to user

    Earner-->>Spender: Approve lease terms (price from PDVF)

    Spender->>Polygon: Call 'createLease()' with terms & payment

    Polygon-->>Spender: Lease Created event with 'leaseId'
```

Spender->>Earner: Send 'leaseId' as proof of payment

Earner->>Polygon: Verify 'leaseId' on-chain

Note over Earner, Spender: Execute Federated Analysis of Features (PySyft)

Earner->>Spender: Send aggregated results

Spender->>Polygon: Finalize lease (optional, can be automated)

Polygon->>Earner: Release funds

Polygon->>Polygon: Record transaction for Royalty & Reputation calculation

4.2. MCP Tool Definitions

To be compliant with the Model Context Protocol, the Pandacea Agent will expose its core functionalities as the following standardized `Tools`.

Tool 1: `request_lease`

- **Description:** "Initiates a request to lease a specific data product for direct access."

Input Schema (JSON):
```
{
 "type": "object",
 "properties": {
  "productId": {
   "type": "string",
   "description": "The unique DID of the data product to be leased."
  },
  "maxPrice": {
   "type": "string",
      "description": "The maximum price the spender is willing to pay, in wei."
    },
    "duration": {
      "type": "string",
      "description": "The requested lease duration (e.g., '24h', '30d')."
    }
  },
  "required": ["productId", "maxPrice", "duration"]
}
```

-

Output Schema (JSON):

```
{
  "type": "object",
  "properties": {
    "status": {
      "type": "string",
      "description": "The status of the lease proposal.",
      "enum": ["pending_approval", "rejected"]
    },
    "leaseProposalId": {
      "type": "string",
      "description": "A unique identifier for this lease proposal."
    }
  }
}
```

-

Tool 2: `request_computation`

- **Description:** "Initiates a request to perform a privacy-preserving computation on a data product using PySyft."

Input Schema (JSON):

```
{
  "type": "object",
  "properties": {
    "productId": {
      "type": "string",
      "description": "The unique DID of the data product to be used."
    },
    "computation": {
      "type": "object",
      "description": "The PySyft computation task to be executed.",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["federated-analysis"]
        },
        "parameters": {
```

```
        "type": "object",
        "description": "Parameters for the computation, e.g., feature and statistic."
      }
    }
  }
 },
 "required": ["productId", "computation"]
}
```

-

Output Schema (JSON):

```
{
 "type": "object",
 "properties": {
  "status": {
   "type": "string",
   "description": "The status of the computation request.",
   "enum": ["pending_approval", "rejected"]
  },
  "computationProposalId": {
   "type": "string",
   "description": "A unique identifier for this computation proposal."
  }
 }
```

}

-

4.3. MyData Agent - WMS Connector Architecture

The integration with Warehouse Management Systems will follow a pragmatic, two-phase approach:

- **Phase 1 (MVP): Direct API Adapter.** For our initial warehouse logistics partner, we will build a custom, point-to-point adapter. This adapter will be a module within the MyData Agent that connects directly to the partner's specific WMS API (e.g., REST, SOAP), authenticates, and transforms their proprietary event data into our canonical LogisticsEventData format. This minimizes initial complexity and accelerates time-to-market.
- **Phase 2 (Post-MVP): Middleware Pattern.** As we onboard more partners, we will refactor the connector architecture to use a middleware pattern. The MyData Agent will communicate with a central, protocol-level service that manages multiple adapters and exposes a single, unified API. This aligns with enterprise best practices and ensures long-term scalability and maintainability.

5. Smart Contract Interfaces (Polygon PoS - MVP)

This section defines the primary functions for the core smart contracts. These are simplified for clarity.

5.1. LeaseAgreement.sol

```
interface ILeaseAgreement {
  // Events
  event LeaseCreated(bytes32 leaseId, address spender, address earner, uint256
price);
  event LeaseApproved(bytes32 leaseId);
  event LeaseExecuted(bytes32 leaseId);

  // Functions
  function createLease(address earner, bytes32 dataProductId, uint256 maxPrice)
external payable;
  function approveLease(bytes32 leaseId) external; // Called by Earner
  function executeLease(bytes32 leaseId) external; // Called by Spender
```

```
    function raiseDispute(bytes32 leaseId, string calldata reason) external;
}
```

5.2. RoyaltyDistributor.sol

```
interface IRoyaltyDistributor {
    // Events
    event RoyaltiesRecorded(bytes32 dataProductId, uint256 amount);
    event RoyaltiesClaimed(address earner, uint256 amount);

    // Functions
    function recordRoyalties(bytes32 leaseId, bytes32 dataProductId) external payable;
    function claimRoyalties() external; // Batched claim for a user
}
```

5.3. Reputation.sol

```
interface IReputation {
    // Functions
    function updateReputation(address user, bool successfulLease) external;
    function getReputation(address user) external view returns (uint256);
}
```

6. Security Considerations (Initial Threat Model)

- **Data Poisoning:**
  - **Threat:** An Earner provides malicious or garbage data to sabotage a Spender's AI model.
  - **Mitigation (MVP):** The Reputation contract will penalize bad actors. The sampleCid in the DataProduct allows for pre-lease verification. The Pandacea Arbitration Court (PAC) provides a mechanism for recourse.
- **Sybil Attacks on DHT:**
  - **Threat:** An attacker creates thousands of fake nodes to pollute the DHT or eclipse a specific user.
  - **Mitigation (MVP):** Libp2p has some built-in resistance. We will require a small PGT stake to publish a DataProduct, making Sybil attacks economically infeasible.
- **Smart Contract Exploits:**

  - **Threat:** Reentrancy attacks, oracle manipulation, etc.
  - **Mitigation (MVP):** Adherence to Checks-Effects-Interactions pattern, use of OpenZeppelin audited base contracts, and a mandatory comprehensive third-party audit before mainnet launch.

7. Resolved Questions & Decisions (v1.2)

- **Q4: Protocol Interoperability:** How does the protocol ensure it can be easily integrated by the growing number of AI agents and LLM applications?
  - **Finding:** The industry is rapidly converging on the Model Context Protocol (MCP) as the standard for agent-tool interaction. A secondary standard for Agent-to-Agent (A2A) communication is emerging for complex collaboration.
  - **Decision:** The protocol will be made MCP-compliant. Each agent will run an MCP Server to expose its data products as `Resources` and its core actions (`request_lease`, `request_computation`) as `Tools`. This ensures any MCP-compatible agent can interact with the Pandacea network out-of-the-box. A future implementation of an A2A protocol will be added to the roadmap to support more advanced, multi-agent negotiation patterns.