Pandacea Protocol - API Specification (v1.1)

Version:	1.1
Status:	**Draft**
Date:	July 12, 2025
Lead Engineer:	Asa Conant
Related Documents:	System Design Document (v1.1)

1. Introduction

This document provides the formal API specification for the Pandacea Protocol MVP. It details the RESTful endpoints exposed by the **MyData Agent** and **Buyer-Side Agent** for agent-to-agent communication over the libp2p network.

1.1. Authentication

All agent-to-agent communication is mutually authenticated using the agents' libp2p peer IDs and associated cryptographic keys. Each request must be signed by the originating agent's private key, and the signature must be included in the request headers.

- **Header:** X-Pandacea-Signature: <signature>

1.2. API Versioning

All API endpoints are versioned to ensure backward compatibility as the protocol evolves. The current version is v1. All endpoint paths must be prefixed with /api/v1.

1.3. Standardized Error Handling

All non-2xx responses will return a standard JSON error object to provide consistent, machine-readable feedback.

- **Standard Error Response Body:**
  ```
  {
    "error": {
      "code": "ERROR_CODE_STRING",
      "message": "A human-readable description of the error.",
      "requestId": "req_123xyz"
  ```

```
  }
}
```

2. Agent API Endpoints

2.1. Data Product Endpoints

GET /api/v1/products

Lists all available data products from an Earner, with support for filtering and pagination. This is the primary endpoint for discovery.

- **Query Parameters:**
  - keywords (string, optional): A comma-separated list of keywords to filter by (e.g., lidar,grasping).
  - limit (integer, optional, default: 20): The maximum number of products to return.
  - cursor (string, optional): The pagination cursor from a previous request to get the next page of results.
- **Response (200 OK):**

```
{
  "data": [
    {
      "productId": "did:pandacea:earner:123/abc-456",
      "name": "Novel Package 3D Scans - Warehouse A",
      "dataType": "RoboticSensorData",
      "keywords": ["robotics", "3d-scan", "lidar"]
    }
  ],
  "nextCursor": "cursor_def456"
}
```

GET /api/v1/products/{productId}

Retrieves the full details for a single data product.

- **Request Parameters:**
  - productId (string, path): The unique identifier of the data product.
- **Response (200 OK):**
  - **Body:** DataProduct schema object (as defined in SDD v1.1).

2.2. Lease Negotiation Endpoints

POST /api/v1/leases

A Spender initiates a lease request to an Earner.

- **Request Body:**

```
{
  "productId": "did:pandacea:earner:123/abc-456",
  "spenderDid": "did:pandacea:spender:789",
  "maxPrice": "50000000000000000", // Price in wei
  "computation": {
    "type": "federated-analysis",
    "parameters": { "feature": "estimatedWeight_g", "statistic": "mean" }
  }
}
```

- **Response (202 Accepted):**

```
{
  "status": "pending_approval",
  "leaseProposalId": "prop-a1b2c3",
  "proposedPrice": "45000000000000000"
}
```

- **Response (403 Forbidden):**

```
{
  "error": {
    "code": "POLICY_REJECTION",
    "message": "Request denied by Earner's policy.",
    "requestId": "req_abc789"
  }
}
```

GET /api/v1/leases/{leaseProposalId}

A Spender polls this endpoint to check the status of a pending lease request.

- **Request Parameters:**
  - leaseProposalId (string, path): The ID from the POST /leases response.
- **Response (200 OK):**

```
{
  "status": "approved", // or "pending_approval", "rejected"
  "leaseProposalId": "prop-a1b2c3",
```

"onChainLeaseId": "0x123abc..." // Included once lease is created on-chain
}

2.3. Data & Computation Endpoints

GET /api/v1/results/{onChainLeaseId}

Retrieves the result of the privacy-preserving computation.

- **Request Parameters:**
  - onChainLeaseId (string, path): The on-chain lease ID.
- **Response (200 OK):**

{
"leaseId": "0x123abc...",
"status": "completed",
"result": { "mean": 845.72 }
}

3. Builder SDK - High-Level Functions

3.1. Spender SDK (Python Example)

```
import pandacea_sdk

spender_agent = pandacea_sdk.init(identity_file="~/.pandacea/spender.json")

# 1. Discover data products with filtering and pagination
data_products = spender_agent.discover(keywords=["lidar", "grasping"], limit=50)

# 2. Initiate a lease
selected_product = data_products[0]
lease_proposal = spender_agent.request_lease(
    product_id=selected_product.id,
    computation_task={
        "type": "federated-analysis",
        "parameters": {"feature": "estimatedWeight_g", "statistic": "mean"}
    }
)

# 3. Approve and execute the lease on-chain.
# This is a high-level, asynchronous function that:
```

```
#   a. Submits the `createLease` transaction to the blockchain.
#   b. Waits for the transaction to be mined.
#   c. Polls the Earner agent until the lease is approved or rejected.
#   d. Handles timeouts and errors gracefully.
lease_result = lease_proposal.execute(max_price=5.0) # Price in PGT

# 4. Get the result
print(f"Federated mean of estimated weight: {lease_result.get('mean')}")
```

3.2. Earner SDK (Python Example)

```
import pandacea_sdk

def my_policy(request):
    if request.spender.is_verified_academic:
        return pandacea_sdk.Approval.APPROVE
    return pandacea_sdk.Approval.REQUIRE_MANUAL

earner_agent = pandacea_sdk.init(
    identity_file="~/.pandacea/earner.json",
    policy_handler=my_policy
)

wms_data_source = earner_agent.add_wms_connector(
    api_endpoint="https://wms.partner.com/api",
    credentials={"api_key": "..."}
)

earner_agent.listen()
```

4. Future Considerations

This section outlines planned improvements for future versions of the API to address the limitations of the MVP.

4.1. Real-time Status Updates

The current polling mechanism (GET /leases/{leaseProposalId}) is inefficient. A future version (v1.2 or v2.0) will support a WebSocket-based connection between agents.

This will allow the Earner to push status updates (approved, rejected) directly to the Spender, eliminating polling and reducing latency.

4.2. Flexible Computation Definitions

The computation object in the POST /leases request is rigid for the MVP. To support a wider range of privacy-preserving tasks, a future version will allow the Spender to provide an IPFS CID pointing to a file that defines a complete PySyft computation plan. This will make the protocol agnostic to the specific computation being performed, increasing its flexibility and power.