Architecture of a Decentralized Data Provision System on Polkadot with ZK, Sidecar, and Tokenomics

Ovcharov Vladimir (c) 2025
Institute of Cybernetics, Kyiv, Ukraine

July 1, 2025

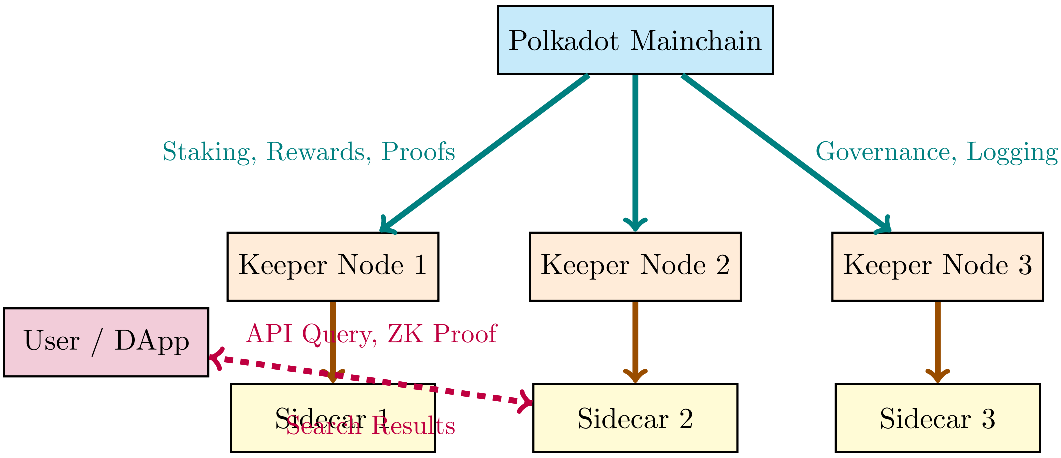
Abstract

This document presents the architecture of a decentralized, privacy-preserving data provision and sharing network. The design leverages a Polkadot-based blockchain, vectorized data sharding, fast Sidecar services, zero-knowledge proofs (ZK, Halo2), and robust tokenomics. It enables secure, permissioned, and reward-driven sharing of information—where data can be searched and consumed, but never fully exfiltrated or leaked.

1 System Overview

- **Polkadot/Substrate Fork**: Mainchain for consensus, staking, reward logic, and on-chain governance.
- **Keeper Nodes**: Sharded, vectorized, encrypted data storage with proof-of-uptime and data delivery.
- **Sidecar Services**: High-speed search, in-memory/disk cache, REST/gRPC API, and ZK proof handling.
- **ZK Modules (Halo2)**: Secure access and privacy, with proofs for all queries.
- **Tokenomics**: Utility token for payment, rewards, staking, and reputation.

2 Architecture Diagram



3 API Specification

3.1 REST/gRPC API Endpoints (Sidecar)

Endpoint	Method	Description
/api/v1/search	POST	Vector similarity search with ZK proof. Returns top-K results and access tokens.
/api/v1/access	POST	Controlled access to data shard; requires ZK proof of authorization.
/api/v1/report	POST	Keeper node reports data served, response times, uptime, ZK activity.
/api/v1/challenge	POST	Exchange storage or uptime challenges for Sybil/data-leak protection.
/api/v1/reward	GET	Returns node rewards, penalties, reputation, and stats.

3.2 Sample JSON Requests/Responses

Listing 1: Sample Vector Search Request

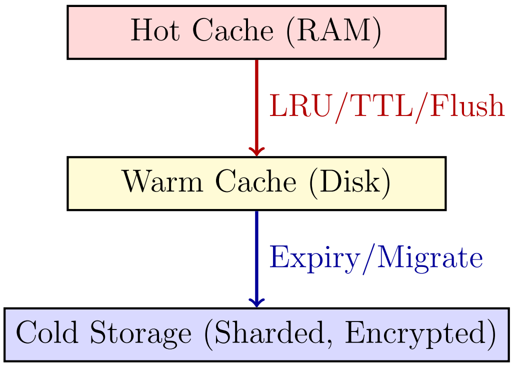
```
POST /api/v1/search
{
  "query_vector": [0.17, 0.32, ...],
  "top_k": 5,
  "zk_proof": "0xabcd1234..."
}
```

Listing 2: Sample Access Response

```
{
  "result_ids": [ "doc1", "doc2", ... ],
  "zk_proof_validated": true,
  "access_tokens": [ "tkn1", "tkn2" ]
}
```

4 Caching Mechanisms

- **Hot Cache**: In-memory (e.g. Redis) for fast, frequent queries and recent ZK proofs.
- **Warm Cache**: Disk-based for less frequent shard/data access.
- **Cold Storage**: Encrypted, sharded datasets; slowest tier.
- **Proof Cache**: Recent validated ZK proofs for API queries.
- **Cache Invalidation**: TTL, LRU, and on data update events.



5 Reward Formula and Tokenomics

5.1 Reward Calculation

$$R = (D_s \cdot w_d + S_r \cdot w_s + P_z \cdot w_z) \cdot U_f \cdot R_m$$

- D_s: Data served (successful queries)
- w_d: Data type weight
- S_r: Response speed score (inverse latency)
- w_s: Speed weight
- P_z: Validated ZK proofs
- w_z: ZK proof weight
- U_f: Uptime factor (0-1)
- R_m: Reputation multiplier (0.5-2.0)

5.2 Reward Distribution Table

Parameter	Typical Range	Description	Source
Data Served (D_s)	0-10000/day	Number of queries served	Sidecar stats
Response Speed (S_r)	1-100	100 = fastest	API logs
Validated Proofs (P_z)	0-1000/day	Number of ZK proofs	ZK engine
Uptime (U_f)	0.9-1.0	Fraction of time online	Node monitor
Reputation (R_m)	0.5-2.0	Node scoring	On-chain history

5.3 Token Economy: Information Market

- **Utility Token (e.g. DATX)**: Payment for queries, rewards, staking.
- **Query Payments**: Users pay tokens to Keeper nodes for access/search. Prices can be dynamic.
- **Rewards**: Distributed per epoch by chain logic (formula above).
- **Slashing**: Misbehavior and Sybil attempts result in penalty and loss of stake.
- **Marketplace**: Third parties and DApps pay for high-value data; rare data = higher reward.

5.4 Sample Use Case: Secure Research Data Provision

A research institute requests access to a medical database:

1. They stake tokens and make a ZK-authenticated request via API.
2. Keeper nodes run vector search and serve only authorized shards.
3. Sidecar caches, logs, and proves all accesses; on-chain for audit.
4. Rewards flow based on quality and quantity of access.

6 Security: Sybil and Data Leak Protection

6.1 Sybil Attack Protection

- **Staking Requirement**: All nodes must lock up tokens to participate.
- **Reputation**: On-chain, community, or KYC-based.
- **Random Challenges**: Proof-of-storage and proof-of-uptime.
- **Rate Limiting**: Prevents spam/flooding by APIs.
- **Slashing**: Penalties for misbehavior.

6.2 Data Leak Prevention

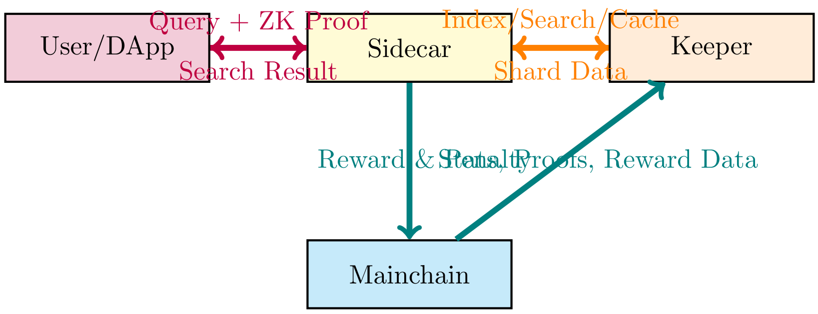
- **Sharded Storage**: No full dataset on any single node.
- **ZK Access Control**: Every access requires a ZK proof.
- **Differential Privacy**: Add noise to aggregated responses.
- **Audit Trail**: All events on-chain for review.
- **Key Rotation**: Periodically change encryption/authorization keys.

7 Step-by-Step Implementation Guide

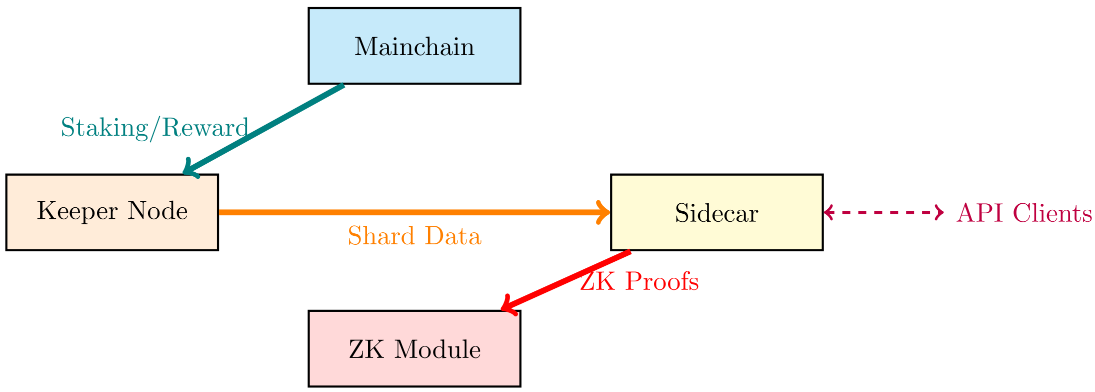
1. **Deploy Mainchain**: Fork Substrate, integrate ZK modules, set up reward logic.
2. **Implement Keeper/Sidecar Nodes**: Sharded encrypted storage, vectorization, REST/gRPC API, caching.
3. **Integrate Tokenomics**: Utility token smart contracts, reward/slash logic.
4. **Connect Clients/DApps**: UI for queries, payments, and ZK proof management.
5. **Test and Audit**: Simulate attacks, test API, run full-stack integration.

8 Diagrams

8.1 Sequence Diagram: Data Query and Reward Flow



8.2 Component Diagram



9 References

- Polkadot/Substrate Docs: `https://substrate.dev/docs`
- Halo2 ZK Framework: `https://zcash.github.io/halo2`
- Vector DB Benchmarks: `https://github.com/ann-benchmarks/ann-benchmarks`
- Research: Privacy-Preserving Data Markets, Decentralized AI