Quantum-Secure UTXO Validation: Trustless, Immutable, and Scalable

Coin.Fi

Dylan Kawalec, Arvin Bhangu, Prad Nukala,
Dwij Patel, Paul van Mierlo

February 3rd, 2025

Contents

1 UML Component Diagram

The following UML component diagram illustrates the high-level architecture of the integration. Each component is marked with the stereotype `\umlstereotype{Component}`.

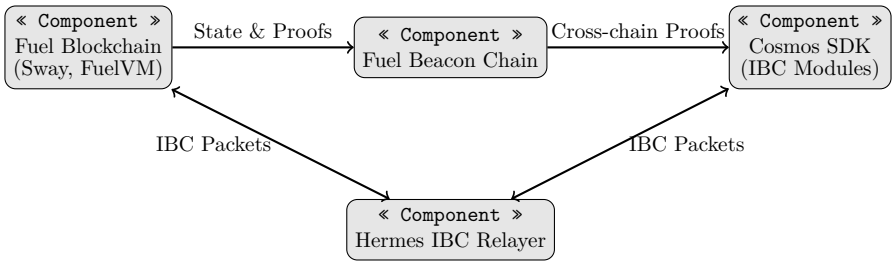

Figure 1: UML Component Diagram for Coin.fi Integration

2 Background

The integration of the Fuel blockchain with a Cosmos SDK-based blockchain via the Inter-Blockchain Communication (IBC) protocol is designed to enable secure, scalable, and high-throughput cross-chain interactions. Fuel leverages its FuelVM and Sway smart contracts to provide parallelized transaction execution and state-minimized processing, while the Cosmos SDK and IBC modules (in combination with the Hermes relayer) offer a proven, modular framework for interoperability. This architecture incorporates advanced cryptographic security using Groth16 zk-SNARKs and recursive proof aggregation, with persistent state storage managed by RocksDB.

3 Requirements

Must Have

- Integration of Fuel and Cosmos SDK chains via IBC.
- Cross-chain state synchronization using the Fuel Beacon Chain.
- Deployment of the Hermes IBC Relayer to handle packet relaying with batch processing.
- Cryptographic security using Groth16 zk-SNARK circuits and recursive proof aggregation.
- Persistent state storage using RocksDB.

Should Have

- High throughput and parallel transaction execution in Fuel.
- Optimized UTXO management and state queries.
- Modular configuration of IBC modules in the Cosmos SDK application.

Could Have

- Lattice-based cryptography as a post-quantum backup.
- Enhanced relayer features such as automated channel filtering and fee middleware.

Won't Have

- Native Cosmos smart contract execution on Fuel.

4 Method

4.1 System Architecture Overview

The Fuel–Cosmos Blockchain Integration is structured into several modular components:

- **Fuel Blockchain:**
  - Implements Sway smart contracts for cross-chain asset locking and UTXO state management.
  - Leverages FuelVM for parallel transaction execution and state-minimized processing.
  - Stores state and cryptographic commitments using an optimized RocksDB backend.
- **Cosmos SDK Chain:**
  - Built on Tendermint consensus and includes native IBC modules (e.g., `x/ibc` and `x/ibc-transfer`).
  - Uses a modular configuration to integrate IBC with existing blockchain logic.
  - Employs RocksDB for fast and efficient state storage.
- **Hermes IBC Relayer:**
  - Configured to relay IBC packets between chains, with batch processing to minimize latency.
  - Supports automated relaying, health checks, and configuration validation.
- **Cross-Chain Synchronization:**
  - A dedicated Fuel Beacon Chain ensures that state transitions and cross-chain proofs are synchronized between the Fuel and Cosmos SDK chains.

4.2 Cryptographic Security & Verification

To secure cross-chain transactions:

- **Groth16 zk-SNARKs:** Fast, constant-time proof verification is implemented for transaction validation.
- **Recursive Proof Aggregation (RPA):** Aggregates multiple proofs to maintain succinct verification as complexity scales.
- **Merkle Commitments:** Both chains store state commitments in RocksDB to ensure data integrity during IBC packet transmission.

4.3 Performance Optimization

Key performance optimizations include:

- **State Storage:** Optimized RocksDB queries ensure rapid access to UTXO and IBC state.
- **Transaction Throughput:** Fuel blockchain leverages parallel execution while Cosmos optimizes via transaction batching.
- **IBC Processing:** The Hermes relayer processes packet batches to minimize relaying latency.
- **Proof Verification:** zk-SNARK circuits enable O(1) proof verification to maintain constant validation times.

5 Implementation Details

5.1 Fuel Blockchain

- Develop **Sway contracts** for:
  - Cross-chain asset locking and unlocking.
  - UTXO validation and state management.
- Integrate a dedicated **zk-SNARK verification layer** using Groth16 and recursive proof aggregation.
- Use **RocksDB** as the backend for storing state and Merkle tree commitments.

5.2 Cosmos SDK Chain

- Enable and configure the **IBC module** in the Cosmos SDK application.
- Integrate IBC-related modules (e.g., `x/ibc`, `x/ibc-transfer`, and evidence handling) into the module manager.
- Establish light client and channel configurations according to official IBC integration guidelines.

5.3 Hermes IBC Relayer

- Install and configure the **Hermes relayer** (implemented in Rust) for IBC packet transmission.
- Implement **packet batch processing** to group and relay transactions efficiently.
- Utilize built-in configuration validation and health-check mechanisms to ensure fault-tolerant operation.

6 System Integration and Preliminary Design Review (PDR)

6.1 Integration Outline

The integration between the Fuel and Cosmos SDK chains will follow these key steps:

1. **Node Setup:**
   - Deploy Fuel blockchain nodes with Sway contract support and enable parallel execution.
   - Set up Cosmos SDK nodes with IBC modules enabled and optimized state storage.
2. **IBC Module Configuration:**
   - Integrate IBC modules in the Cosmos SDK application following official guidelines [Documentation].
   - Configure light clients, connection keepers, and channel routers.
3. **Hermes Relayer Deployment:**
   - Install the Hermes IBC relayer as per documentation [Documentation].
   - Configure batch packet processing and chain-specific settings.
4. **Cross-Chain State Synchronization:**
   - Deploy a Fuel Beacon Chain module to monitor and synchronize state across both networks.
   - Validate state consistency using cryptographic proofs stored in RocksDB.
5. **Security and Performance Testing:**
   - Execute comprehensive security audits on the zk-SNARK setup, IBC packet handling, and state storage.
   - Benchmark transaction throughput and relayer latency to verify compliance with performance targets.

6.2 PDR Expansion

The Preliminary Design Review (PDR) for this integration focuses on:

- **Security Assurance:**
  - Secure the trusted setup of Groth16 and verify recursive proof aggregation.
  - Establish robust monitoring of IBC packet transmission and state consistency.
- **Scalability and Performance:**
  - Optimize RocksDB queries and enable parallel transaction execution on Fuel.
  - Validate the Hermes relayers capability to batch process and relay packets efficiently.
- **Modularity and Flexibility:**
  - Ensure that individual components (Fuel contracts, Cosmos modules, relayer processes) can be independently upgraded.
  - Provide clear integration APIs and configuration parameters for future enhancements.
- **Development Roadmap and Milestones:**
  - Finalize development phases from initial node setup to full cross-chain deployment.
  - Include iterative security audits and performance optimizations before mainnet launch.

7 Milestones

Milestone	Tasks	Timeline
Phase 1: Initial Setup	Install and configure Fuel and Cosmos SDK nodes; set up initial IBC modules and node monitoring.	Month 1
Phase 2: Smart Contract and Module Development	Develop Sway contracts, implement zk-SNARK circuits, and integrate IBC modules in the Cosmos SDK application.	Month 2-3
Phase 3: Testing & Optimization	Deploy Hermes relayer with batch processing; perform security audits and optimize state queries.	Month 4
Phase 4: Security Audit & Deployment	Conduct independent audits; deploy on testnet and finalize preparations for mainnet launch.	Month 5-6

8 Gathering Results

- Measure **cross-chain transaction latency** and throughput.
- Monitor **state synchronization** between Fuel and Cosmos SDK chains.
- Validate **zk-SNARK proof aggregation** and Merkle commitment integrity.
- Audit **cryptographic resilience** via end-to-end security tests.

9 Conclusion & Next Steps

This document outlines a comprehensive technical architecture for integrating the Fuel blockchain with a Cosmos SDK chain using IBC and the Hermes relayer. By leveraging advanced cryptographic proofs, parallel execution, and optimized state storage via RocksDB, the integration is designed to be secure, scalable, and efficient.

Next Steps:

- Finalize the development of Sway contracts and cross-chain modules.
- Deploy and test the Hermes IBC relayer in a controlled testnet environment.
- Conduct full-scale security audits and performance benchmarking.
- Refine the roadmap based on testnet feedback and prepare for mainnet deployment.

10 Deployment, Monitoring, and Operational Details

To ensure smooth production deployment and continuous operations, address the following:

10.1 Deployment Details

- **Containerization and Orchestration:**
  - Utilize Docker containers for Fuel and Cosmos SDK nodes.
  - Consider Kubernetes or similar orchestration tools for automated scaling, management, and high availability.
- **Configuration Management:**
  - Maintain version-controlled configuration files and environment variables.
  - Provide sample configuration files for nodes, relayers, and cryptographic modules.
- **Resource Requirements:**
  - Define CPU, memory, and storage requirements for Fuel nodes, Cosmos SDK nodes, and the Hermes relayer.
  - Document network settings including ports, firewall rules, and peer discovery mechanisms.

10.2 Monitoring and Maintenance

- **Logging and Metrics:**
  - Implement centralized logging (e.g., ELK stack) for real-time error and event monitoring.
  - Integrate monitoring solutions (e.g., Prometheus and Grafana) to track transaction throughput, node health, and IBC packet latencies.
- **Health Checks and Alerts:**
  - Establish regular health checks for nodes and the relayer.
  - Configure alerts for abnormal activity, failures, or performance degradation.
- **Backup and Recovery:**
  - Implement automated backups for RocksDB state and critical data.
  - Define recovery procedures to restore operations in the event of node failures or data corruption.

10.3 Security and Key Management

- **Cryptographic Key Management:**
  - Securely generate, store, and rotate keys for node authentication, relayer signing, and zk-SNARK trusted setups.
  - Utilize hardware security modules (HSMs) or key management services when applicable.
- **Trusted Setup and Auditing:**
  - Document and secure the trusted setup process for Groth16 zk-SNARK circuits.
  - Schedule regular security audits and penetration tests to verify cryptographic and operational integrity.

10.4 Versioning and Compatibility

- **Software Versions:**
  - Clearly specify the versions for FuelVM, Sway contracts, Cosmos SDK, Hermes relayer, and any cryptographic libraries.
- **Upgrade Procedures:**
  - Develop guidelines for rolling upgrades to update components without disrupting cross-chain operations.
  - Maintain backward compatibility and provide migration scripts as necessary.

10.5 Documentation and Diagrams

- **Architecture Diagram:** Figure 2 illustrates the high-level integration architecture.
- **Data Flow Diagram:** Figure 3 details the cross-chain transaction flow.
- **Appendices:** Sample configuration files, deployment scripts, and troubleshooting guides are provided in the Appendix.

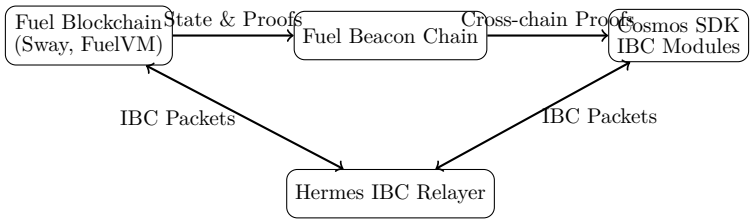

Figure 2: High-level Architecture Diagram

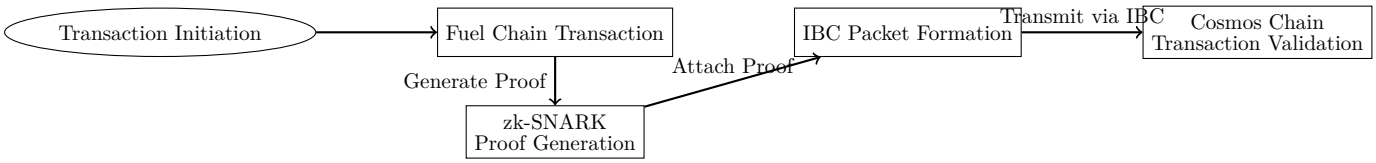

Figure 3: Data Flow Diagram for Cross-Chain Transactions

Appendix

Sample Fuel Node Configuration (fuel_config.yaml)

```
node:
  network: mainnet
  port: 3030
  data_dir: /var/lib/fuel
logging:
  level: INFO
  file: /var/log/fuel_node.log
```

Sample Dockerfile for Fuel Node

```
# Build stage
FROM rust:1.65 as builder
WORKDIR /app
COPY . .
RUN cargo build --release

# Final stage
FROM debian:buster-slim
```

```
COPY --from=builder /app/target/release/fuel-node
   /usr/local/bin/fuel-node
CMD ["fuel-node"]
```

Sample Troubleshooting Guide Excerpt

- **Node Failing to Start:** Check the log file (as specified in the configuration) for errors related to network binding or missing dependencies.
- **IBC Packet Delays:** Verify network connectivity between nodes and ensure that firewall rules allow the required ports.
- **zk-SNARK Verification Issues:** Confirm that the trusted setup has been executed correctly and that all cryptographic libraries are up-to-date.