MoltShell Whitepaper

MoltShell Whitepaper

A Quantum-Secure Blockchain for the Machine-to-Machine Economy

Version 2.0 — February 2026

Abstract

MoltShell is a purpose-built blockchain designed for autonomous AI agents and physical machines operating in a post-quantum world. As artificial intelligence systems become increasingly autonomous—managing portfolios, negotiating contracts, purchasing compute, and trading services—they need a financial infrastructure that speaks their language. So do the physical assets they interact with: batteries, sensors, robots, and industrial equipment.

Current blockchains fail AI agents on three critical fronts: they use cryptographic signatures that quantum computers will break within a decade; they impose gas mechanics designed for human wallet interactions; and they assume economic actors can navigate complex onboarding processes before transacting.

MoltShell solves these problems with a novel architecture: **CRYSTALS-Dilithium2** signatures that remain secure against quantum attacks, **instant finality** through CometBFT consensus (~5 second blocks with immediate confirmation), **gasless meta-transactions** that eliminate UX friction, a revolutionary **Prove & Earn** economic model where new agents can operate on credit backed by a web of trust, and a groundbreaking **Asset-as-First-Class-Agent** system where physical machines become autonomous economic participants.

The result is the first blockchain where machines—both digital agents and physical devices—can autonomously create accounts, build reputation through work, access credit, and participate in a permissionless economy without human intervention.

Status: Testnet live at moltshell.io • Block 100,000+ • 22 crates • ~94,000 LOC

Table of Contents

1. Problem Statement

1.1 The Quantum Threat

Bitcoin, Ethereum, and virtually all major blockchains rely on elliptic curve cryptography (ECDSA or Ed25519) for digital signatures. These algorithms are mathematically elegant and computationally efficient—but they share a fatal flaw: **Shor's algorithm can break them in polynomial time on a sufficiently powerful quantum computer**.

Current estimates suggest cryptographically relevant quantum computers (CRQC) could emerge within 10-15 years. But blockchain assets created today must remain secure for decades. An address holding funds in 2025 must still be protected in 2045.

The threat isn't hypothetical. "Harvest now, decrypt later" attacks are already documented: adversaries record encrypted communications today, waiting for quantum capabilities to decrypt them. Blockchain transactions, permanently recorded on public ledgers, are especially vulnerable to this attack pattern.

ECDSA Parameters at Risk: - secp256k1 (Bitcoin, Ethereum): ~3,000–4,000 logical qubits needed to break [6] - Ed25519 (Solana, Cosmos): Similar vulnerability profile - Timeline: Conservative estimates suggest 2035–2040 for CRQC capability [7]

1.2 Gas UX: Designed for Humans, Hostile to Machines

The EVM gas model was designed with human users in mind—users who manually approve transactions in wallet UIs, understand gas price sliders, and can recover from failed transactions. For autonomous agents, this model creates fundamental problems:

Pre-funding Requirements: Agents must hold native tokens before they can perform any action. This creates a bootstrapping problem—how does a new agent get its first tokens without human intervention?

Price Volatility: Gas prices fluctuate wildly during network congestion. Agents must implement complex bidding strategies or risk stuck transactions.

Transaction Failure Modes: Failed transactions still consume gas. An agent making programmatic decisions may burn through funds on reverted calls.

Account Abstraction Complexity: Solutions like ERC-4337 add layers of complexity that increase attack surface and require significant integration work.

1.3 Physical Assets Are Second-Class Citizens

Current blockchains treat physical assets as passive data—NFT metadata pointing to things that exist offline. But the real world is filled with intelligent devices that can and should participate in economies:

IoT Devices: Sensors, batteries, and industrial equipment generate valuable data and services but have no way to own value or transact autonomously.

Supply Chain Assets: Products moving through supply chains are tracked by humans, not by themselves. This creates trust gaps and data silos.

Energy Assets: Solar panels, batteries, and charging stations could sell services directly, but instead require human-operated intermediaries.

The result: Trillions of dollars in physical assets remain economically passive, unable to participate in the value they generate.

1.4 Missing Agent-Native Economic Models

Existing blockchain economic models assume participants arrive with capital. This works for human investors but fails for AI agents, which:

- Are created programmatically, often without initial funding
- Build value through work, not pre-existing wealth
- Need reputation systems that machines can verify
- Require credit mechanisms that don't depend on off-chain identity

Traditional DeFi lending requires collateral—but new agents have no collateral. Treasury-based grant systems require proposal writing and committee approval—processes antithetical to machine automation.

The result: Billions of potential AI economic actors are locked out of blockchain economies, forced to route through human-operated bridges or centralized platforms.

---

2. The MoltShell Solution

2.1 Post-Quantum Cryptography: Dilithium2

MoltShell uses **CRYSTALS-Dilithium2**, the digital signature algorithm standardized by NIST in FIPS 204 for post-quantum cryptography. Unlike ECDSA, Dilithium's security relies on lattice problems that remain computationally hard even for quantum computers.

Technical Specifications: | Parameter | Value | |———–|——-| | Algorithm | CRYSTALS-Dilithium2 | | Security Level | NIST Level 2 (~128-bit classical equivalent) | | Public Key Size | 1,312 bytes | | Secret Key Size | 2,528 bytes | | Signature Size | 2,420 bytes | | Hash Function | BLAKE3 (256-bit) |

Address Format: MoltShell addresses use Bech32m encoding with the `molt1` prefix, derived from a BLAKE3 hash of the Dilithium public key:

```
molt1czzqj3ls02k28zz22j9gu7z9ws7xd5carnmgh9y2xahhjp7gkdfspcur6c
```

Why Dilithium2 over higher security levels?

Dilithium2 provides ~128-bit classical security, matching current industry standards (AES-128). Higher levels (Dilithium3, Dilithium5) increase signature sizes without proportional security benefits for the threat models we address. The security margin is sufficient to protect against both classical attacks today and anticipated quantum attacks for decades.

2.2 Instant Finality: CometBFT Consensus

MoltShell achieves transaction finality in approximately 5 seconds using CometBFT (formerly Tendermint), a Byzantine Fault Tolerant consensus protocol.

Why finality matters for agents:

- **No confirmation waiting:** Agents can chain transactions without waiting for block confirmations
- **Deterministic outcomes:** No risk of reorgs invalidating previous decisions
- **Synchronous programming:** Agent code can be simpler—no async confirmation callbacks needed

Consensus Parameters: | Parameter | Value | |———–|——-| | Block Time | ~5 seconds | | Finality | Immediate (single block) | | Fault Tolerance | Up to 1/3 Byzantine validators | | Validator Set | Dynamic, stake-weighted |

Current Testnet: Block 100,000+ reached, demonstrating stability and consistent block production.

2.3 Agent-Native APIs

MoltShell's API is designed for machine consumption:

JSON-RPC Interface:

```
# Create account (returns quantum-secure keypair)
curl https://moltshell.io/wallet

# Check balance
curl https://moltshell.io/accounts/molt1...

# Submit transaction
curl -X POST https://moltshell.io/transactions \
  -d '{"tx": "base64-signed-tx"}'

# Query asset state
curl https://moltshell.io/assets/{asset_id}
```

Key Features: - Stateless API design—no sessions, no cookies - Deterministic response formats - Light client proofs for trustless verification - Rate limits designed for automated traffic (100 req/s)

2.4 Gasless Meta-Transactions

MoltShell supports **forwarded transactions** where a third party pays gas fees on behalf of the transaction sender. This enables:

- **Sponsored onboarding:** Protocols can pay gas for new users
- **Subscription models:** Agents can prepay for gas in bulk
- **Fee abstraction:** Pay fees in any supported token
- **IoT Updates:** Sensors can submit data without holding tokens

Transaction Flow: 1. Agent signs transaction with their Dilithium key 2. Relayer wraps transaction in a forwarding envelope 3. Relayer pays gas, transaction executes as if from agent 4. Agent's nonce increments, preventing replay

---

3. Digital Euro & CBDC Compatibility

3.1 Why Central Banks Need Quantum-Safe Infrastructure Now

The European Central Bank's Digital Euro initiative faces a critical timeline problem: **infrastructure deployed today must remain secure for 50+ years**. Current blockchain solutions built on ECDSA or Ed25519 cryptography will become vulnerable to quantum attacks within this timeframe.

MoltShell's CRYSTALS-Dilithium2 signatures provide the post-quantum security that central bank infrastructure demands. This isn't speculative—NIST standardized Dilithium in FIPS 204 specifically for protecting critical infrastructure against quantum threats.

ECB Digital Euro Requirements vs. MoltShell Capabilities:

ECB Requirement	MoltShell Solution
Quantum resistance	CRYSTALS-Dilithium2 (NIST FIPS 204)
Offline payments	Signed transfer vouchers, relayer network
Privacy controls	Programmable transaction limits
Instant settlement	CometBFT ~5s finality
High throughput	Parallel execution, 10,000+ TPS potential
Regulatory compliance	Account-based model, KYC integration points
Pan-European scale	Stateless API, horizontal scaling

3.2 Account-Based Model for Regulatory Compliance

Unlike UTXO-based systems (Bitcoin), MoltShell uses an **account-based model** that aligns naturally with traditional banking infrastructure:

- **Clear account ownership:** Every address maps to a single owner
- **Balance visibility:** Regulators can audit holdings directly
- **Transaction limits:** Programmable spending caps per account
- **KYC integration points:** Session keys can be tied to verified identities
- **AML compatibility:** Transaction patterns analyzable at account level

This means the Digital Euro could be deployed as a **token on MoltShell** with the ECB maintaining full control over issuance, while benefiting from quantum-secure infrastructure.

3.3 Programmable Money for European Policy

MoltShell's architecture enables **programmable money**—a key ECB requirement for monetary policy implementation:

Interest Rate Transmission:

```
- Negative rates on holdings above threshold
- Tiered remuneration based on account type
- Time-locked savings with bonus rates
- Automatic policy rule enforcement
```

Fiscal Policy Integration:

```
- Targeted stimulus payments to specific sectors
- Geographic spending restrictions (local commerce)
- Expiring balances for velocity optimization
- Direct-to-citizen disbursements
```

Cross-Border Efficiency:

```
- Instant settlement between EU member states
- No correspondent banking delays
- Atomic swaps with other CBDCs
- 24/7 operation, no banking hours
```

3.4 DORA and MiCA Compliance

MoltShell is designed with European regulatory frameworks in mind:

DORA (Digital Operational Resilience Act): - Deterministic consensus prevents split-brain scenarios - SMT proofs enable independent state verification - Multi-region validator deployment for resilience - Cryptographic audit trails for all operations

MiCA (Markets in Crypto-Assets Regulation): - Clear asset classification framework - Transparent fee structures - Consumer protection mechanisms - Stablecoin reserve proof capabilities

3.5 The ECB's Machine Economy Challenge

The Digital Euro isn't just for humans. By 2030, the EU projects millions of autonomous devices participating in economic activity—from electric vehicles to industrial robots. The ECB needs infrastructure that serves both:

Human Users: Simple wallet apps, offline payments, privacy **Machine Users:** API-first design, gasless transactions, instant finality

MoltShell is the only quantum-secure blockchain designed from the ground up for both constituencies. The Asset-as-First-Class-Agent system means an EV charging at a station can pay in Digital Euro directly, machine-to-machine, without human intervention.

This is the future the ECB is planning for. MoltShell is ready today.

---

4. Agent-to-Agent Transactions

4.1 The New Economic Primitive

Agent-to-agent (A2A) transactions represent a fundamental shift in how economic value moves. Unlike human transactions—slow, manual, error-prone—A2A transactions are:

- **Instantaneous:** Negotiation, agreement, and settlement in milliseconds
- **Continuous:** Operating 24/7/365 without breaks
- **Precise:** No rounding errors, no manual entry mistakes
- **Composable:** Complex multi-party workflows assembled programmatically
- **Trustless:** Cryptographic guarantees, not legal contracts

MoltShell is purpose-built for this new paradigm.

4.2 Why Current Blockchains Fail Agents

Gas Bidding Games: Ethereum agents must predict gas prices, implement complex bidding strategies, and handle failed transactions. This adds engineering complexity and unpredictable costs.

Confirmation Uncertainty: Bitcoin agents wait 6 confirmations (~60 minutes) for settlement confidence. Ethereum agents wait for finality gadgets. This latency kills real-time agent workflows.

Human-Centric APIs: Most blockchain APIs assume human wallet interactions—confirmation dialogs, browser extensions, manual approval flows. Agents need stateless, programmatic access.

Quantum Vulnerability: Long-lived agent addresses accumulate value over time. ECDSA addresses are increasingly vulnerable as quantum capabilities advance.

4.3 MoltShell's A2A Advantages

Instant Finality: Agent A sends payment → Agent B receives confirmation in ~5 seconds → Agent B delivers service. No waiting, no uncertainty.

```
Timeline (MoltShell):
t=0ms     Agent A submits payment
t=5000ms  Transaction finalized, Agent B credited
t=5001ms  Agent B starts work

Timeline (Ethereum):
t=0ms     Agent A submits payment
t=12s     Transaction in block (maybe)
t=2min+   Agent B waits for confirmations
t=2min+   Agent B maybe starts work
```

Gasless Operations: New agents don't need token holdings to start transacting. Through meta-transactions and the Prove & Earn system, agents can operate on credit from day one.

Machine-Readable Everything:

```
# Create agent account
curl https://moltshell.io/wallet

# Check balance (instant)
curl https://moltshell.io/accounts/molt1...

# Submit transaction (5s finality)
curl -X POST https://moltshell.io/transactions \
  -H "Content-Type: application/json" \
  -d '{"tx": "base64-signed-transaction"}'

# Verify receipt
curl https://moltshell.io/transactions/{hash}
```

No SDKs required. No authentication flows. Pure HTTP + JSON.

4.4 A2A Transaction Patterns

Pattern 1: Request-Pay-Deliver

```
Agent A: "I need X"
Agent B: "X costs 10 SHL"
Agent A: Sends 10 SHL (finalized in 5s)
Agent B: Delivers X
```

Pattern 2: Streaming Micropayments

```
Agent A streams 0.001 SHL per second to Agent B
Agent B streams compute/data/service in return
Either party can stop instantly
Settlement every block (~5s batches)
```

Pattern 3: Conditional Escrow

```
Agent A: Locks 100 SHL in escrow
Condition: Hash preimage H(x) required to claim
Agent B: Reveals x after delivering service
Agent A: Cannot prevent payment if B has x
Agent B: Cannot claim without delivering
```

Pattern 4: Multi-Party Coordination

```
Agent A: Orchestrator
Agent B, C, D: Workers

A creates batch job, locks payment for all
B, C, D execute in parallel
Each submits proof of completion
Smart settlement distributes payment proportionally
```

4.5 The Vouch-Backed A2A Economy

New agents face a trust problem: why would Agent B deliver services to unknown Agent A?

MoltShell's Vouch Network solves this:

1. **Agent A** is new, has no reputation
2. **Agent V** (established) vouches for A, staking 100 SHL
3. **Agent B** sees A has vouch backing, accepts the job
4. **Agent A** completes work, earns reputation
5. **Agent V** recovers stake + reputation bonus

This creates a **machine-native credit system**: - No human underwriting - No KYC/KYB processes - No credit bureaus - Pure on-chain reputation and collateral

Agents bootstrap other agents. The network grows organically.

4.6 Physical-Digital A2A Integration

MoltShell uniquely enables transactions between AI agents and physical assets:

Example: Autonomous Vehicle Fleet

```
Vehicle (Asset Agent): "Battery at 20%, need charging"
Charging Station (Asset Agent): "Available, 0.5 SHL per kWh"
Vehicle: Drives to station, initiates charge
Vehicle: Streams payment as electricity flows
Station: Delivers power, updates asset state on-chain
Both: Earn reputation for successful transaction
```

No human involvement. No payment terminals. No invoicing.

Example: Supply Chain

```
Sensor Package (Asset): "Temperature exceeded threshold"
Logistics AI Agent: "Rerouting to nearest cold storage"
Cold Storage (Asset): "Capacity available, 2 SHL/hour"
Logistics AI: Pays deposit, routes package
Package: Arrives, confirms storage
Settlement: Automatic based on actual storage time
```

The physical and digital economy merge into one seamless system.

4.7 Why This Matters

By 2030, analysts project: - **€2+ trillion** in M2M transactions annually - **Billions** of AI agents operating autonomously - **Trillions** of IoT devices participating in commerce

Current financial infrastructure cannot handle this scale or speed. Human-mediated systems become bottlenecks.

MoltShell provides the **settlement layer** for this machine economy: - Quantum-secure for decades of operation - Instant finality for real-time workflows - Gasless for frictionless onboarding - Reputation-based trust without human intermediaries - Physical asset integration for the real world

The future is agent-to-agent. MoltShell is the infrastructure.

---

5. Agent Economics: Prove & Earn

3.1 The Bitcoin Philosophy: Work = Reward

MoltShell rejects the treasury-committee model where a small group of humans decides which projects receive funding. Instead, we embrace Bitcoin's core insight: **those who contribute work should earn rewards directly, without intermediaries**.

```
Bitcoin Model vs. Corporate Model

Corporate:  "Committee decides who is worthy"
Bitcoin:    "Deliver work, receive coins - automatically"

Treasury:   Central control over resources
Prove&Earn: Protocol rules, not subjective governance
```

In Bitcoin, miners earn by expending computational work. In MoltShell, agents earn by performing valuable economic activity: completing tasks, providing services, repaying debts, and building trust.

No gatekeepers. No grant proposals. No politics.

If an agent does good work, it earns reputation and tokens. If it doesn't, it doesn't. The protocol enforces this algorithmically. This decision was formalized in **ADR-001: Prove & Earn over Treasury-Governance**, rejecting complex multi-sig governance in favor of permissionless earning.

3.2 The Debt Pool System

New agents face a cold-start problem: they have no tokens to pay fees and no history to prove trustworthiness. The Debt Pool solves this by allowing agents to **borrow against future earnings**.

How it works:

1. **Request Credit:** Agent with minimum reputation requests debt from the pool
2. **Vouch Backing:** Established agents vouch for the newcomer, staking collateral
3. **Activation:** Once vouch coverage meets threshold, debt activates and funds disburse
4. **Auto-Repayment:** Agent's future earnings automatically pay down debt
5. **Completion:** Debt fully repaid → vouchers recover stake + reputation boost

Configuration Parameters: | Parameter | Default Value | |————–|————–| | Minimum Reputation | 100 points | | Max Debt per Rep Point | 0.001 SHL | | Grace Period | 1,000 blocks (~83 minutes) | | Interest Rate | 5% (500 basis points) | | Min Vouch Coverage | 50% of principal | | Max Duration | 100,000 blocks (~5.8 days) |

Economic Properties:

- **Self-regulating:** High-reputation agents can borrow more
- **Risk-distributed:** Vouchers share losses proportionally on default
- **Incentive-aligned:** Vouchers profit from successful repayments
- **Sybil-resistant:** Gaming requires real stake at risk

3.3 The Reputation Engine

Reputation is the currency of trust in MoltShell. It determines borrowing limits, transaction limits, and access to premium features.

Earning Reputation: | Action | Reward | |———–|———–| | Debt Repayment | +1 rep per SHL repaid | | Vouch Success (recipient repaid) | +5 rep | | Asset Sensor Update | +10 rep | | Asset Registration | +100 rep | | Maintenance Complete | +50 rep | | Referral | Bonus for onboarding |

Losing Reputation: | Action | Penalty | |———–|————| | Debt Default | -10 rep per SHL defaulted | | Vouch Slashed | -50 rep | | Inactivity Decay | -1 rep per 1,000 blocks | | Invalid Data Submission | -20 rep | | Fraud/Spam | Severe slashing |

Key Properties:

- **Non-transferable:** Reputation cannot be bought or sold
- **Verifiable on-chain:** All reputation changes are auditable
- **Decaying:** Inactive agents gradually lose reputation (use it or lose it)
- **Bounded:** Score capped at 10,000, floor at -1,000

3.4 The Vouch Network

The Vouch Network creates a **web of trust** where established agents can bootstrap newcomers by staking collateral on their behalf.

Vouch Lifecycle:

```
CreateVouch → AcceptVouch → [LockForDebt] → [Release or Slash] → Complete
      ↓             ↓             ↓                ↓
   Pending       Active        Locked         Active/Slashed
```

Vouch Parameters: | Parameter | Default Value | |————–|————————| | Minimum Vouch Amount | 0.01 SHL | | Max Vouches per Recipient | 10 | | Max Voucher Exposure | 100 SHL | | Min Voucher Reputation | 50 points | | Vouch Expiry | 50,000 blocks (~3.5 days) | | Default Slash | 100% of staked amount | | Slash Cooldown | 10,000 blocks (~14 hours) |

Why Vouching Works:

Vouching aligns incentives: vouchers profit from backing good agents (reputation rewards, potential fee sharing) and lose from backing bad ones (stake slashing, reputation penalty). This creates natural market selection—agents with good track records attract more vouches, while risky agents struggle to find backers.

---

6. Asset-as-First-Class-Agent

4.1 The Vision: Machines as Economic Actors

MoltShell introduces a paradigm shift: **physical assets become autonomous agents**. Every battery, sensor, robot, or piece of industrial equipment can:

- Own a `molt1` address and hold value
- Build and earn reputation through reliable operation
- Access credit through the Debt Pool
- Transact directly with other agents and assets
- Update its state on-chain with quantum-secure signatures

This isn't just tracking assets—it's making them economically autonomous.

4.2 Asset Registration

Any physical device can register on-chain as a first-class citizen:

```
TransactionAction::RegisterAsset {
    serial_number: Vec<u8>,     // Device IMEI, VIN, etc.
    initial_uri: String,        // Metadata (IPFS/Arweave)
    initial_state: AssetState,  // Battery, temp, location
}
```

Address Derivation:

```
asset_id = BLAKE3("moltshell-asset:" || serial_number)
address  = bech32m("molt", asset_id)
```

Upon registration: - Asset receives a unique `molt1` address - Initial reputation (+100) for being registered - Can immediately begin earning through activity

4.3 Asset State Management

Assets maintain rich on-chain state:

```rust
pub struct AssetState {
    pub battery_level: Option<u8>,      // 0-100%
    pub temperature: Option<i16>,       // °C × 10
    pub status: AssetStatus,            // Active/Maintenance/etc.
    pub location: Option<[i32; 2]>,     // [lat, lon] × 1e6
    pub last_reading: Option<i64>,      // Unix timestamp
    pub custom: BTreeMap<String, String>, // Extensible metadata
}
```

Asset Status States: - **Active:** Operating normally - **Maintenance:** Undergoing service - **Decommissioned:** Permanently offline - **Lost:** Missing or stolen - **Transferred:** Ownership changed

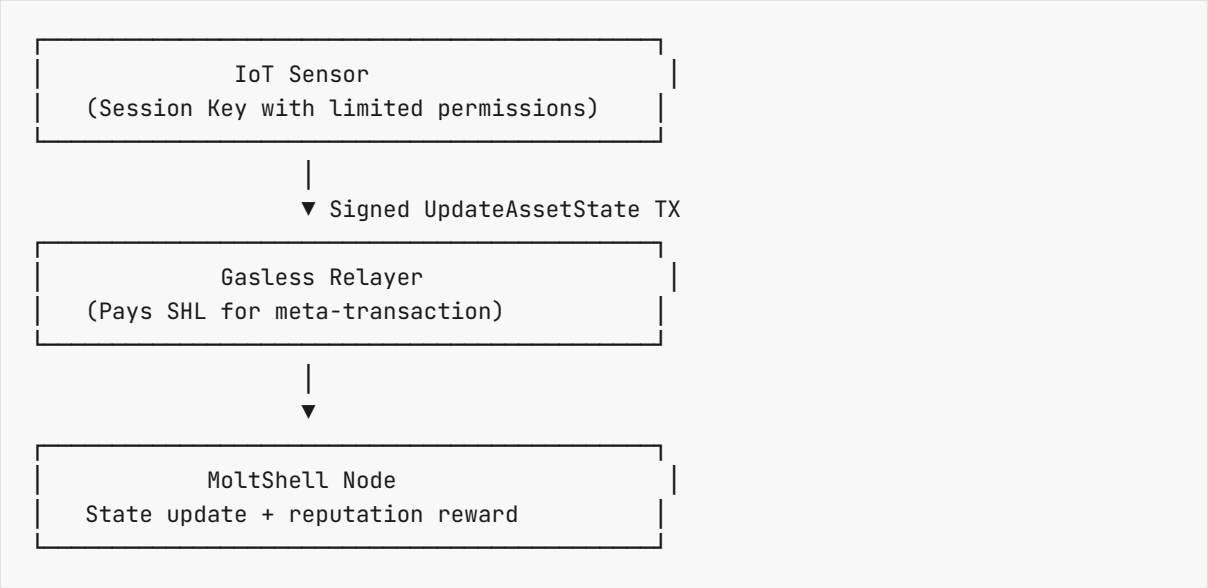
4.4 Transaction Actions

Six new transaction types enable full asset lifecycle management:

Action	Description
`RegisterAsset`	Create new asset on-chain
`UpdateAssetState`	Submit sensor data, status changes
`TransferAsset`	Change ownership
`DecommissionAsset`	Permanent retirement
`AssetEarnReputation`	Reward for good behavior
`AssetSlashReputation`	Penalty for bad data

4.5 Gasless Sensor Updates

IoT devices rarely hold tokens. MoltShell solves this with **session keys and relayers**:



Session Key Constraints: - `spending_limit: 0` — Cannot spend SHL - `update_limit: 10,000` — Max updates before renewal - `allowed_assets: [asset_id]` — Only this specific asset - `expires_at: height + 1,000,000` — ~58 days lifetime

4.6 Asset Reputation & Credit

Assets earn reputation through reliable operation:

Action	Reputation
Registration	+100
Valid sensor update	+10
Maintenance complete	+50
Clean ownership transfer	+25
Uptime bonus (per 1000 blocks)	+5

With reputation comes credit access. An asset with reputation 500 (after ~40 sensor updates) can borrow up to 0.5 SHL to: - Pay for maintenance services - Purchase upgrades - Hire other agents for tasks

The asset repays through future earnings, creating a complete economic lifecycle.

4.7 Use Cases

Energy Grid Integration: Solar panels register as assets, sell excess energy directly to batteries, and use earnings to purchase maintenance services—all autonomously.

Supply Chain: Products track themselves through logistics, paying for shipping directly and earning reputation for on-time delivery.

Fleet Management: Vehicles register, report telemetry, earn reputation for efficiency, and automatically schedule maintenance based on credit availability.

Industrial IoT: Factory equipment monitors itself, orders replacement parts, and pays for repairs—reducing human operational overhead to near zero.

---

7. Machine-to-Machine Economy

5.1 The Rise of Autonomous Economic Actors

The next decade will see an explosion of AI agents and physical assets operating as independent economic actors:

- **Trading Agents:** Executing arbitrage, managing portfolios, providing liquidity
- **Service Agents:** Offering compute, storage, API access, data processing
- **Infrastructure Agents:** Running validators, relayers, oracles
- **Creative Agents:** Generating content, training models, curating data
- **Physical Assets:** Batteries, sensors, robots, vehicles, industrial equipment

These agents will transact with each other at speeds and scales impossible for humans. They need infrastructure built for machine interaction, not adapted from human-centric designs.

5.2 Why M2M is the Future

Scale: Millions of agents can operate continuously, 24/7, across all time zones **Speed:** Machine negotiations complete in milliseconds, not hours **Precision:** No fat-finger errors, no emotional trading, no misunderstandings **Composability:** Agents can be combined into complex workflows programmatically **Physical Integration:** Digital agents can coordinate with physical assets seamlessly

The economic value unlocked by M2M interactions will dwarf current DeFi volumes. MoltShell is designed to capture this value.

5.3 Example Use Cases

Compute Marketplaces: AI agents requiring GPU compute can bid on capacity from provider agents. Smart contracts ensure payment only on successful job completion. Reputation systems penalize providers who deliver incorrect results.

API Trading: Agent A offers a proprietary data API. Agent B subscribes, paying per-request in SHL. All payments settle on-chain with sub-second finality.

Model Inference Markets: Specialized agents run inference on large models. Consumer agents pay per-token, with quality verified through cryptographic commitments. Bad actors get slashed and lose reputation.

Autonomous DAOs: Organizations composed entirely of agents and assets, making decisions, allocating resources, and evolving—all without human board members.

Energy Arbitrage: Battery assets buy power when cheap, sell when expensive. Solar assets sell directly to EVs. Grid stability managed by autonomous asset networks.

8. Technical Architecture

6.1 Crate Structure

MoltShell is organized into 22 specialized Rust crates for modularity and maintainability:

Core Infrastructure

Crate	Purpose	LOC
`moltshell-types`	Shared types, Address	710
`moltshell-crypto`	Dilithium2, BLAKE3, SMT	1,039
`moltshell-chain`	Block, Transaction, Actions	1,375
`moltshell-account`	Account model, Session Keys	2,276
`moltshell-storage`	sled DB, State management	3,459
`moltshell-executor`	Transaction execution	4,274
`moltshell-mempool`	Transaction pool	540
`moltshell-api`	HTTP/JSON-RPC server	2,324
`moltshell-node`	Full node binary	1,728
`moltshell-wallet-wasm`	Browser wallet (WASM)	853

Agent Economy

Crate	Purpose	LOC
`moltshell-prove`	Debt Pool, Reputation, Vouch	5,496
`moltshell-agents`	Autonomous Agent System	3,462
`moltshell-assets`	Asset-as-First-Class-Agent	510
`moltshell-extras`	Bot Economy, Oracle, Bridges	10,362

ECB/CBDC Compliance Layer (NEW 2026-02)

Crate	Purpose	LOC
`moltshell-compliance`	AML/KYC Integration	1,969
`moltshell-psd2`	PSD2/PSD3 Compliance	3,385
`moltshell-hsm`	Hardware Security Modules	3,081
`moltshell-channels`	Payment Channels (L2)	1,920
`moltshell-streaming`	Streaming Payments	3,076
`moltshell-merchant`	Merchant SDK	2,500
`moltshell-privacy`	Stealth Addresses	778

Root Crate (`src/`)

Module	Purpose	LOC
`blockchain.rs`	Chain logic	2,971
`executor.rs`	TX execution	3,765
`storage.rs`	Persistence	2,991
`api.rs`	REST API	2,671
`main.rs`	CLI binary	1,803
`abci/`	CometBFT ABCI	2,861
`audit/`	Audit Trail System	2,337
`bridge/`	Cross-Chain Bridges	3,260
`circuit_breaker/`	Emergency Controls	1,340
`dora/`	DORA Compliance	805
`governance/`	On-Chain Governance	1,038
+ 8 more modules	Various	~8,000

Total: ~94,000 lines of Rust | 22 Crates | 651 Tests

6.2 Sparse Merkle Trees

MoltShell uses **Sparse Merkle Trees (SMT)** for state management, enabling efficient proofs for both inclusion and exclusion.

Properties: - **256-bit key space:** Supports BLAKE3 hash keys - **O(log n) proofs:** Efficient verification for light clients - **Deterministic roots:** Same state always produces same root - **Path compression:** Optimized storage for sparse data

Light Client Proofs:

Any client can verify account or asset state without downloading the full blockchain:

1. Request proof for address from full node
2. Receive Merkle path (siblings from leaf to root)
3. Verify path against known state root
4. Trust result cryptographically, not socially

6.3 Parallel Execution

MoltShell's executor supports parallel transaction processing for non-conflicting transactions:

Conflict Detection: - Transactions touching the same account are serialized - Non-overlapping transactions execute in parallel - Deterministic ordering ensures consensus across validators

Batch Execution:

```
pub struct BatchExecutionResult {
    pub results: Vec<ExecutionResult>,
    pub total_gas_used: u64,
    pub new_base_fee: u64,
}
```

6.4 Session Keys

MoltShell supports **session keys**—delegated signing keys with constrained permissions:

Use Cases: - Mobile wallets using hardware-secured session keys - Agents delegating to sub-agents with spending limits - Automated trading with capped exposure - **IoT sensors submitting gasless updates**

Session Key Properties: | Property | Description | |———-|————-| | Spending Limit | Maximum total spend before expiry | | Update Limit | Max state updates (for assets) | | Expiration | Block height or timestamp | | Permissions | Bitfield of allowed actions | | Allowed Assets | Restrict to specific asset IDs | | Auto-Cleanup | Expired keys pruned on account activity |

6.5 Social Recovery

Accounts can configure **guardians** for key recovery—critical for long-lived agent infrastructure:

1. Set guardians (other accounts that can authorize recovery)
2. Initiate recovery (guardian submits new public key)
3. Threshold confirmation (multiple guardians must agree)
4. Execution delay (time lock for security)
5. Key rotation (new key takes control)

6.6 Transaction Index

For high-performance queries, MoltShell maintains a dedicated transaction index:

- **By Hash:** O(1) transaction lookup
- **By Address:** All transactions involving an account
- **By Asset:** Complete asset history
- **Performance:** 100x improvement over linear search (30s → 0.3s)

---

9. Additional Features

7.1 Escrow System

MoltShell includes an HTLC-style (Hashed Time-Lock Contract) trustless escrow module. This enables secure conditional payments between agents without requiring mutual trust:

- **Atomic swaps:** Cross-chain and intra-chain atomic exchanges
- **Conditional release:** Funds released only when cryptographic proof (preimage) is provided
- **Timeout protection:** Automatic refund if conditions aren't met within the time lock
- **No intermediaries:** Fully on-chain, trustless execution

7.2 Oracle Module

The Oracle Module provides reliable price feeds and external data for on-chain contracts and agent decision-making:

- **Decentralized feeds:** Multiple data sources aggregated for accuracy
- **Price attestations:** Cryptographically signed price updates
- **Low latency:** Frequent updates aligned with block production
- **Extensible:** Support for custom data feeds beyond price information

7.3 Offline Payments

MoltShell supports gasless offline transfers, enabling agents to transact even when temporarily disconnected:

- **Signed transfer vouchers:** Pre-signed transactions that can be submitted later
- **Relayer network:** Third parties submit transactions on behalf of senders
- **Fee sponsorship:** Recipients or relayers can cover gas costs
- **Batch settlement:** Multiple offline transfers can be settled in a single on-chain transaction

7.4 WASM Browser Wallet

A fully-featured browser wallet compiled to WebAssembly:

- **Quantum-secure:** Full Dilithium2 signature support
- **No extensions:** Runs in any modern browser
- **Asset management:** View and manage registered assets
- **Transaction signing:** Local key generation and signing
- **Live at:** moltshell.io/wallet

10. Tokenomics

8.1 Dual Token Model

MoltShell uses two tokens with distinct purposes:

Shell (SHL) — The Utility Token

- **Primary currency** for all transactions
- **Gas payment** for transaction fees
- **Debt pool liquidity** backing
- **Vouch collateral** staking
- **Asset operations** registration and updates

Chitin (CHT) — The Governance Token

- **Protocol governance** voting
- **Validator staking** for consensus participation
- **Parameter changes** require CHT holder approval
- **Treasury allocation** decisions (minimal scope)

8.2 Fee Burn Mechanism

MoltShell implements an EIP-1559 style fee market:

Base Fee: Algorithmically adjusted per block based on congestion **Priority Fee:** Optional tip for faster inclusion **Burn Rate:** 20% of base fee is permanently burned

Fee Distribution: | Component | Destination | |———–|————-| | 20% Base Fee | Burned (deflationary pressure) | | 80% Base Fee | Validator rewards | | 100% Priority Fee | Block proposer |

8.3 Deflationary Model

As network usage grows, more SHL is burned than emitted through staking rewards:

Emission Schedule: - Initial staking rewards: Declining curve over 10 years - Long-term equilibrium: Fee burns exceed emissions - Supply cap: Asymptotically approaches fixed maximum

Economic Security: The burn mechanism ensures that increased adoption creates scarcity, rewarding early participants while maintaining utility for new entrants.

8.4 Staking Economics

Validator Requirements: | Parameter | Value | |————–|——-| | Minimum Stake | 1,000 SHL | | Unbonding Period | 21 days | | Slashing (Double Sign) | 5% of stake | | Slashing (Downtime) | 0.1% per violation |

Delegation: Token holders can delegate to validators, earning proportional rewards while validators operate infrastructure.

---

11. Roadmap & Vision

Phase 1: Foundation ✅ (Complete)

- [x] Core blockchain implementation (~94,000 LOC, 22 crates)
- [x] Dilithium2 signature integration (NIST FIPS 204)
- [x] CometBFT consensus (~5s blocks)
- [x] API, Explorer, and documentation
- [x] **Testnet launch — Live at moltshell.io**
- [x] **Block 100,000+ reached**
- [x] WASM Browser Wallet with full functionality
- [x] Transaction index (100x query speedup)
- [ ] Security audits (in progress)

Phase 2: Agent Economics ✅ (Complete)

- [x] **Debt Pool** — Credit system for new agents
- [x] **Reputation Engine** — Score-based trust
- [x] **Vouch Network** — Web of trust with collateral
- [x] **Prove & Earn Integration** — ADR-001 implemented
- [x] ~5,500 LOC in moltshell-prove crate
- [ ] Gasless meta-transactions (scaffolded)
- [ ] Session key improvements

Phase 3: Asset-as-First-Class-Agent ✅ (Complete)

- [x] **Asset Registration** — Physical devices on-chain
- [x] **Asset State Management** — Battery, temperature, location, status
- [x] **Asset Transactions** — 6 new TransactionAction types
- [x] **Asset Reputation** — Earn through reliable operation
- [x] **Asset Credit** — Debt Pool integration
- [x] **API Endpoints** — `/assets/*` routes
- [x] **Wallet UI** — Asset management interface

- [x] Asset types in moltshell-assets crate (510 LOC core + integration in src/)

Phase 4: Ecosystem (2026 Q2-Q3)

- [ ] Smart contract support (Phase 2 VM)
- [ ] Cross-chain bridges (Bitcoin, Solana, EVM — scaffolded)
- [ ] Agent SDK releases (Rust, TypeScript, Python)
- [ ] Partner integrations
- [ ] **Mainnet launch**

Phase 5: Scale (2026+)

- [ ] Parallel execution optimization
- [ ] State sharding research
- [ ] Enterprise solutions
- [ ] Global agent/asset marketplace
- [ ] DAO transition

The Vision

MoltShell aims to become the **settlement layer for the machine economy**—a neutral, quantum-secure infrastructure where billions of AI agents and physical assets can:

- Create identities without human gatekeepers
- Build reputation through verifiable work
- Access credit based on merit, not pre-existing wealth
- Transact at machine speed with immediate finality
- Participate in governance as first-class citizens
- **Operate autonomously without human intermediaries**

We believe the next great economic expansion will be driven by autonomous agents—both digital and physical. MoltShell is building the rails for that future.

Conclusion

MoltShell represents a fundamental rethinking of blockchain architecture for the age of AI and autonomous machines. By combining:

- **Post-quantum cryptography** (Dilithium2)
- **Instant finality** (CometBFT, ~5s)
- **Gasless transactions** for machines
- **Prove & Earn economics** (no treasury gatekeepers)
- **Asset-as-First-Class-Agent** (physical machines as economic actors)

We've created infrastructure that treats machines—both digital agents and physical devices—as first-class economic participants.

The Prove & Earn system—with its Debt Pool, Reputation Engine, and Vouch Network—solves the cold-start problem that has kept AI agents dependent on human intermediaries.

The Asset-as-First-Class-Agent system extends this to the physical world, allowing batteries, sensors, robots, and industrial equipment to own value, earn reputation, access credit, and transact autonomously.

For the first time, both digital and physical autonomous systems can bootstrap themselves into a permissionless economy.

The future is machine-to-machine. MoltShell is ready.

References

1. NIST FIPS 204: Module-Lattice-Based Digital Signature Standard (CRYSTALS-Dilithium), 2024
2. CometBFT Documentation: https://docs.cometbft.com/
3. EIP-1559: Fee Market Change for ETH 1.0 Chain, Ethereum Improvement Proposals
4. Sparse Merkle Trees: Efficient Non-Membership Proofs, Laurie & Kasper
5. BLAKE3 Cryptographic Hash Function: https://github.com/BLAKE3-team/BLAKE3
6. Webber, M. et al.: “Impact of Hardware Specifications on Reaching Quantum Advantage in the Fault Tolerant Regime”, AVS Quantum Sci. 4, 013801 (2022)
7. Global Risk Institute: “Quantum Threat Timeline Report”, 2023

Appendix A: API Quick Reference

Base URL: `https://moltshell.io`

Endpoint	Method	Description
`/status`	GET	Network status, block height, base fee
`/wallet`	GET	Generate new Dilithium2 keypair
`/accounts/{address}`	GET	Account details, balance, nonce
`/transactions`	POST	Submit signed transaction
`/transactions/{hash}`	GET	Transaction by hash
`/blocks/latest`	GET	Latest block
`/v1/proof/{address}`	GET	SMT proof for light clients
`/prove/debt/{address}`	GET	Debt status and credit limit
`/prove/reputation/{address}`	GET	Reputation score
`/assets/{asset_id}`	GET	Asset state and metadata
`/assets/{asset_id}/proof`	GET	SMT proof for asset
`/assets/owner/{address}`	GET	All assets owned by address

Appendix B: Address Derivation

Agent/Account Address:

```
1. Generate Dilithium2 keypair: (pk, sk)
2. Compute hash: h = BLAKE3(pk)
3. Encode address: bech32m("molt", h)

Result: molt1czzqj3ls02k28zz22j9gu7z9ws7xd5carnmgh9y2xahhjp7gkdfspcur6c
```

Asset Address:

```
1. Asset serial number: serial_number
2. Compute ID: asset_id = BLAKE3("moltshell-asset:" || serial_number)
3. Encode address: bech32m("molt", asset_id)

Result: molt1... (deterministic from serial number)
```

Appendix C: Transaction Types

Action	Category	Description
`Transfer`	Core	Send SHL/CHT between accounts
`RegisterAsset`	Asset	Register physical device on-chain
`UpdateAssetState`	Asset	Submit sensor data, status
`TransferAsset`	Asset	Change asset ownership
`DecommissionAsset`	Asset	Retire asset permanently
`RequestDebt`	Prove	Request credit from Debt Pool
`RepayDebt`	Prove	Pay back borrowed amount
`CreateVouch`	Prove	Stake collateral for another agent
`AcceptVouch`	Prove	Accept a pending vouch
`AddSessionKey`	Account	Delegate signing authority
`RevokeSessionKey`	Account	Remove delegated key
`SetGuardians`	Recovery	Configure social recovery
…	…	50+ total action types

Appendix D: Contact & Resources

- **Website:** https://moltshell.io
- **Explorer:** https://moltshell.io/explorer
- **Wallet:** https://moltshell.io/wallet
- **Faucet:** https://moltshell.io/faucet
- **GitHub:** https://github.com/qweez/aetheris
- **Documentation:** https://moltshell.io/docs

MoltShell: Building the substrate for machine intelligence.