

ChainEquity

Tokenized Security Prototype with Compliance Gating

Background

Cap-table management, equity issuance, and secondary settlements for private companies remain painful—manual spreadsheets, slow transfer agents, and limited liquidity.

Tokenization on programmable blockchains offers a path forward: instant settlement, transparent ownership records, and automated compliance checks. But most "security token" platforms are black-box SaaS solutions that hide the core mechanics.

Your challenge: Build a working prototype showing how tokenized securities could function on-chain with compliance gating, corporate actions, and operator workflows—without making regulatory claims.

Project Overview

Individual or small team project with no strict deadline.

Deliverables:

- Gated token contract (allowlist-based transfer restrictions)
- Issuer service for wallet approval and token minting
- Event indexer producing cap-table snapshots
- Corporate action system (splits, symbol changes)
- Operator UI or CLI for demonstrations
- Test suite and gas benchmarks
- Technical writeup with decision rationale

Core Requirements

1. Gated Token Contract

Create a token representing synthetic equity with transfer restrictions:

- Standard token interface (ERC-20, SPL Token, or equivalent)
- Allowlist mechanism: only approved wallets can send/receive
- Transfer validation: check sender AND recipient allowlist status
- Revert transfers if either party not approved
- Emit events for all transfers and approvals
- Owner/admin controls for allowlist management

Example flow:

- Wallet A requests approval → Admin approves → Wallet A added to allowlist
- Wallet B not approved
- Transfer from A to B → FAILS (B not on allowlist)
- Admin approves B
- Transfer from A to B → SUCCESS (both on allowlist)

2. Issuer Service (Off-Chain)

Build a minimal backend for operator workflows:

- Approve/deny wallet addresses (KYC mock)
- Submit allowlist updates to contract
- Mint tokens to approved wallets

- Query allowlist status
- Trigger corporate actions

Implementation options:

- Node.js/Python backend with web3 library
- CLI tool with commands for each operation
- Simple admin dashboard (minimal UI)

3. Event Indexing & Cap-Table Export

Index blockchain events to produce ownership records:

- Listen for Transfer, Mint, Burn events
- Maintain current balance per wallet
- Generate "as-of block" snapshots
- Export cap-table in CSV/JSON format
- Include: wallet address, balance, percentage ownership
- Query historical cap-table at any block height

Example cap-table output:

Wallet	Balance	Ownership %
0x1234...5678	10,000	50%
0xabcd...ef01	7,000	35%
0x9876...4321	3,000	15%

4. Corporate Actions (Required: 2 Types)

Implement mechanisms to modify token state:

Action 1: Stock Split (7-for-1)

Requirements:

- Multiply all balances by 7
- Maintain proportional ownership (percentages unchanged)
- Update total supply accordingly
- Emit event documenting the split
- Example: Wallet with 2 shares → 14 shares after split

Implementation approaches:

- Option A: Iterate through all holders and update balances on-chain (gas intensive)
- Option B: Deploy new contract with multiplied supply, migrate holders
- Option C: Keep balances unchanged but adjust display logic (virtual split)
- Document your choice and tradeoffs

Action 2: Symbol/Ticker Change

Requirements:

- Change token symbol/ticker (e.g., "OLD" → "NEW")
- Preserve all balances and ownership
- Update metadata visible to explorers/wallets

- Emit event documenting the change
- Example: Wallets holding "ACME" now show "ACMEX"

Implementation approaches:

- Option A: Update contract metadata (if mutable)
- Option B: Deploy new contract with new symbol, migrate balances
- Option C: Wrapper contract that proxies to original
- Document your choice and tradeoffs

5. Operator Demo (UI or CLI)

Build a demonstration showing:

- Mint tokens to approved wallet → SUCCESS
- Transfer between two approved wallets → SUCCESS
- Transfer to non-approved wallet → BLOCKED
- Approve new wallet → Transfer now succeeds
- Execute 7-for-1 split → Balances multiply by 7
- Change ticker symbol → Symbol updates, balances unchanged
- Export cap-table at specific block

Format options:

- CLI with commands: approve-wallet, mint, transfer, split, change-symbol, export-cap-table
- Simple web UI with buttons for each operation
- Scripted demo (JavaScript/Python) that runs automatically

Technical Architecture

Chain Selection

Choose a blockchain and justify your decision:

- Ethereum (Sepolia testnet): Most mature ecosystem, ERC-20 standard, high gas costs
- Polygon (Mumbai testnet): EVM-compatible, low fees, fast finality
- Solana (Devnet): High throughput, low cost, different programming model (Rust)
- Arbitrum/Optimism: L2 with lower fees, EVM-compatible
- Local devnet: Ganache, Hardhat Network, Anvil, or solana-test-validator

Requirements:

- Must be movable to public chain (even if you develop locally)
- Document deployment addresses and transaction hashes
- Include reproducible setup scripts
- If using testnet, never use real funds

Smart Contract Stack

Ethereum/EVM options:

- Solidity with Hardhat or Foundry
- OpenZeppelin contracts for ERC-20 base
- Custom allowlist logic on top

Solana options:

- Rust with Anchor framework
- SPL Token program as base
- Custom program for transfer hooks

Backend/Indexer

Suggested tools:

- Node.js: ethers.js, web3.js, or @solana/web3.js
- Python: web3.py or solana-py
- Event listening: WebSocket subscriptions or polling
- Storage: In-memory, SQLite, or PostgreSQL for cap-table

Code Quality Requirements

- Clean, readable code with clear separation (contracts / backend / indexer)
- One-command setup: make dev, docker-compose up, or equivalent
- Concise README with setup, deployment, and demo instructions
- Test suite: unit tests for contracts, integration tests for flows
- Deterministic demo scripts that can be run repeatedly
- Decision log documenting key architectural choices
- Gas report for all contract operations
- Environment variables for secrets (.env.example provided)

Success Criteria

Your project will be evaluated on these metrics:

Category	Metric	Target
Correctness	False-positive transfers (non-allowlisted)	0
Correctness	False-negative blocks (allowlisted)	0
Operability	"As-of block" cap-table export	Generated successfully
Corporate Actions	Split and symbol change both work	Demonstrated
Performance	Transfer confirmation time	Within testnet norms
Performance	Indexer produces cap-table	<10s after finality
Documentation	Chain/standard rationale documented	Clear and justified

Additional Requirements:

- Admin safety controls prevent unauthorized minting/approval
- Gas report shows reasonable costs for all operations
- Test coverage for happy path and failure scenarios
- Reproducible setup (anyone can run your demo)
- Risks/limitations documented (no false compliance claims)

Required Test Scenarios

Your test suite must cover:

- Approve wallet → Mint tokens → Verify balance
- Transfer between two approved wallets → SUCCESS
- Transfer from approved to non-approved → FAIL
- Transfer from non-approved to approved → FAIL
- Revoke approval → Previously approved wallet can no longer receive
- Execute 7-for-1 split → All balances multiply by 7, total supply updates
- Change symbol → Metadata updates, balances unchanged
- Export cap-table at block N → Verify accuracy
- Export cap-table at block N+10 → Verify changes reflected
- Unauthorized wallet attempts admin action → FAIL

Gas Benchmarks (EVM Chains)

Provide gas costs for key operations:

Operation	Target Gas
Mint tokens	<100k gas
Approve wallet	<50k gas
Transfer (gated)	<100k gas
Revoke approval	<50k gas
Stock split (per holder)	Document actual cost
Symbol change	<50k gas

If costs exceed targets, document why and propose optimizations.

Submission Requirements

Submit the following:

- Code repository (GitHub preferred)
- Brief technical writeup (1-2 pages) with:
 - - Chain selection rationale
 - - Corporate action implementation approach
 - - Key architectural decisions
 - - Known limitations and risks
- Demo video or live presentation showing all test scenarios
- Documentation of AI tools and prompts used
- Gas report and performance metrics
- Test results (pass/fail for all scenarios)
- Deployment addresses (if using testnet)
- Reproducible setup scripts

Important Constraints

Avoid turnkey frameworks:

Don't use pre-built "security token" platforms (Polymath, Harbor, etc.). Build the core mechanics yourself to demonstrate understanding. You may use standard libraries (OpenZeppelin) but implement gating logic yourself.

No compliance claims:

This is a technical prototype. Include a disclaimer that this is NOT regulatory-compliant and should not be used for real securities without legal review.

Secrets management:

Use environment variables for private keys, RPC URLs, etc. Provide `.env.example`. Never commit secrets.

Testnet only:

If deploying to public networks, use testnets/devnets only. Never use real funds.

Technical Contact

For questions or clarifications:

Bryce Harris - bharris@peak6.com

Build Strategy

Recommended phases:

- Setup: Choose chain, set up dev environment, deploy basic token
- Core Gating: Implement allowlist and transfer restrictions
- Issuer Service: Build admin backend for approvals and minting
- Indexer: Listen to events, generate cap-table snapshots
- Corporate Actions: Implement split and symbol change
- Testing: Write comprehensive test suite
- Demo: Build operator UI/CLI and create demo video
- Documentation: Write technical report and decision log

Optional Hard-Mode Add-Ons

If you finish core requirements and want more challenge:

- Multi-sig admin controls (require N of M signatures for sensitive operations)
- Vesting schedules with cliff and linear unlock
- Partial transfer restrictions (max daily volume per wallet)
- Dividend distribution mechanism
- Secondary market with order book (on-chain DEX)
- Cross-chain bridge for token migration
- Privacy features using zero-knowledge proofs
- Upgradeable contracts with proxy pattern
- Gas optimization (aim for 50% reduction)
- On-chain governance for parameter changes

Final Note

This project demonstrates how blockchain primitives can streamline equity management. The goal is to understand the mechanics, not to build a production system.

Core principles:

- Transparency: All transfers are auditable on-chain
- Automation: Compliance checks happen programmatically
- Efficiency: Settlement is instant vs. T+2 traditional
- Accuracy: Cap-table is always correct and queryable

Document your tradeoffs. Every design decision has pros and cons—explain yours clearly.