synzkhub

SynZK Hub — Zero-Knowledge Action & Synchronization Layer

Whitepaper • Version 1.3 (Detailed • Compact Layout)

Abstract

SynZK Hub is a privacy-first action layer that enables verifiable, privacy-preserving operations across multiple blockchains. This compact full-length edition retains the depth of v1.1 while significantly reducing whitespace. It covers formal action definitions, circuit composition and versioning, hybrid proving systems and aggregation, cross-chain verification, private EVM rollup design, economic incentives, security model, attestations, performance targets, SDKs, and governance.

Table of Contents

1. Problem Statement & Goals

Public blockchains expose transaction metadata (sender, amounts, counterparts), enabling unwanted profiling and MEV. Existing privacy systems often silo assets, require heavy UX, or lack cross-chain verification. SynZK Hub aims to make private actions composable and verifiable on any chain without trusted bridges.

- Privacy by default with cryptographic correctness; reveal only what must be verified.
- Composable with DeFi, NFTs, governance, and agent systems via modular proof modules.
- Cross-chain by design using proof-of-validity and canonical adapters, not trusted multisigs.

- Predictable fees and developer-first SDKs; explicit circuit registry and versioning.
- Upgrade paths anchored by semantic compatibility hashes and DAO governance.

2. System Model & Terminology

We consider users, DApps, agents, relayers, provers, verifiers and settlement chains.

```
Formalization (informal):
- intent I: public statement (dst, asset, class of op, limits, deadline)
- constraints C: predicates over I (policy proofs, rate limits, allowlists)
- witness W: private inputs (amounts, secrets, attributes, randomness)
- circuit F: verifies (I, C, W) ⇒ True
- proof π = Prove(F, (I, C), W); Verify(F, (I, C), π) ⇒ True
```

3. Protocol Overview

- Build intent I; choose module set M from registry.
- Generate proofs π■ per module; optionally aggregate to Π.
- Verify Π on source chain; emit commitment event.
- Sync Engine relays (domain_src, domain_dst, intent_hash, Π, nonce, expiry).
- Destination verifier checks Π; state transition executes; receipts returned.

4. ZK Action Layer (Circuits, Proof Modules)

Proof Modules encapsulate circuits+verifiers with metadata for safe composition. Selectors compose modules; a semantic hash ensures compatibility. Developers publish modules to a registry with fees and versioning governed by the DAO.

Field	Description
module_id	Unique identifier in registry
semver	x.y.z; major bumps deploy new verifier
compat_hash	Hash of semantic signature (inputs/outputs/constraints)
verifier	On-chain address implementing Verify(intent, proof)
params	Curve/SRS commitments; domain separators; aux data

Composition

```
Selector S builds composite constraints S(M1..Mk).
Aggregate proof Π proves all Mi hold for the same intent I.
Adapters expose a uniform Verify(I, Π) to destination chains.
```

Features

- Circuit registry & versioning; semantic checks at compose-time.
- Confidential amounts and shielded addresses built-in.
- Batch/recursive aggregation to reduce verification cost.
- Deterministic verifiers with on-chain selectors.

5. Proof Systems & Aggregation

We target a hybrid approach: zk-SNARKs for efficiency and zk-STARKs for transparent setup. Aggregation via recursion, batch verification, or pairing multi-proof.

Aspect	SNARK	STARK
Setup	Structured (SRS)	Transparent
Proof size	Small	Larger
Verifier cost	Low	Moderate
Latency	Low	Moderate
Security	DL assumptions	Hash-based (FRI)

Aggregation & Targets

• Proof size: ≤ 50–150 kB aggregated • EVM verification: ≤ 250k–600k gas • Throughput: ≥ 100 actions/s with pooled proving on GPU clusters.

6. Sync Engine & Cross-Chain Verification

The Sync Engine is a stateless relayer set with staking, slashing, and reputation. Safety stems from on-chain verification; liveness from redundancy. Replay protection uses nonces and domain separators; time-boxed expiries bound message validity.

```
Message m := (domain_src, domain_dst, intent_hash, Π, nonce, expiry)
- Nonce/domain prevent cross-replay
- Expiry bounds message validity
```

```
- Destination verifier checks Π against registered module set
```
- No trusted multisigs; proof-of-validity on destination chain.
- Optional light-clients for finality; challenge windows for optimistic parts.
- Relayer staking with slashing for malformed messages or equivocation.

7. SynZK Rollup (Private EVM)

A private EVM-compatible rollup executes shielded transactions and commits state roots to settlement chains. ZK■WASM adapters support non■EVM circuits. MEV is reduced via encrypted mempool and batch sequencing.

State Model

```
State = (Accounts, Notes, Nullifiers)
Spend: proves ownership of notes and outputs new commitments.
Nullifier set prevents double-spends; commitments hide amounts/owners.
```

8. Wallet / Identity & Account Abstraction

SynZK Wallet SDK issues ephemeral identities bound to ZK policies. Account Abstraction (4337-style) enables batched actions and sponsored fees.

- Ephemeral keys with user-defined linkability windows.
- Rate-limit and compliance proofs without revealing PII.
- Plugins for MPC/HW wallets/guardians; social recovery via threshold proofs.

9. ZK-AI Connector

Agents commit to plan hashes and provide policy-compliance proofs (e.g., asset lists, VaR bounds) while keeping prompts/weights private. The connector mediates intent generation and on-chain verification.

```
Agent Flow:
1) Compute plan P; publish h = H(P)
2) Produce policy proof π_policy (e.g., VaR ≤ τ, whitelisted assets)
3) Submit (h, π_policy, intent I) → Verify → Execute
```

10. Economic Design & Token ($SYNZK)

$SYNZK powers fees, staking, governance, and rewards. Fees include proof-generation subsidy pools, verification gas, and relayer tips.

Fee Model (illustrative)

```
User fee f = α·v + β·g + γ·p
v: value class; g: on-chain verify gas; p: proving units
(α,β,γ) set by DAO per module tier.
```

Allocation (illustrative)

Category	Share	Vesting
Community & Rewards	35%	48m emission
Core Contributors	20%	48m linear, 12m cliff
Ecosystem Fund	20%	DAO grants
Investors/Backers	15%	36m linear
Treasury & Liquidity	10%	DAO governed

Incentives

- Relayer/Prover staking yields funded by fees.
- Module authors share registry revenue (per-call split).
- Slashing funds redistributed to honest actors.

11. Validator/Relayer Roles & Slashing

Relayers/provers stake $SYNZK; misbehavior (malformed messages, equivocation, SLA breaches) is penalized with graduated slashes.

```
Slashing (examples):
- Malformed message / invalid Π → σ1
- Double-relay conflicting payloads → σ2
- Downtime > δ (SLA violation) → σ3
Evidence on-chain enables trustless challenges.
```

12. Security Model, Threats, & Audits

Layered defenses across circuits, verifiers, relayers, and rollup logic. Independent audits, continuous verification pipelines, public bug bounty.

Threat	Vector	Mitigation
Circuit bug	Constraint omission	Formal specs, audits, test vectors
Replay	Cross-domain reuse	Nonce + domain separators
Forgery	Weak params	Modern ZK assumptions; parameter hygiene
MEV leakage	Order leakage	Encrypted mempool; batch sequencing
Relayer collusion	Censorship	Multiple relayers; liveness penalties

13. Compliance-Friendly Privacy (Attestations)

Anonymous credentials (e.g., BBS+) let issuers provide attestations proven in ZK (age ≥ threshold, region whitelists, sanctions exclusion). No PII on-chain.

14. Performance & Benchmarks (Targets)

Metric	Target (Testnet)
Proof generation (single action)	< 500 ms (GPU) / < 2 s (CPU)
Aggregated verification gas (EVM)	≤ 400k for 4 actions
Relayer end■to■end latency	< 5 s cross■chain (optimistic)
Rollup throughput	≥ 50 TPS private transfers (batch)

15. SDKs & Reference Interfaces

```
// TypeScript SDK (illustrative)
const intent: Intent = {
  op: "swap", dst: "0xTokenB", chainDst: "base",
  amountClass: "≤1000_USD_equiv", deadline: now()+300
}
const modules = [confAmount(), rateLimit(1000), kycRegion("allowed")]
const Π = await client.prove(intent, modules)
await client.submit(intent, Π, { dstChain: "base" })
```

```
// Solidity (illustrative)
interface IVerifier {
  function verify(bytes calldata intent, bytes calldata proof) external view returns (bool);
}
contract Selector {
  function verify(bytes calldata intent, bytes calldata aggProof) external view returns (bool) {
    // dispatch aggregated proof Π to module verifiers
  }
}
```

16. Deployment, Upgrades, & Governance

Deploy on Sepolia, BSC Testnet, Base Sepolia, Polygon Amoy, Solana devnet. Upgrades use semver with on-chain registry. Governance: token-holder DAO with security council for emergency patches and parameter freezes.

17. Roadmap & Testnet Plan

- T-0: Circuit registry + two modules (conf. transfer, rate-limit).
- T-1: Aggregation + Selector; EVM verifier adapters.
- T-2: Sync Engine relayer set (staking + minimal slashing).
- T-3: Private rollup MVP; encrypted mempool; batch proving.
- T-4: Wallet SDK alpha; ZK-AI connector POC.
- T-5: Public testnet incentives, audits, security competitions.

18. Risk Disclosures

Cryptographic systems carry inherent risks, including novel attack vectors. Economic parameters are illustrative and may change via DAO governance. This document is not financial advice and does not constitute an offer of securities.

19. Glossary

Action, Aggregation, Attestation, Circuit, Intent, Nullifier, Proof Module, Selector, Verifier, ZK■WASM.