Backstop

Proof-Triggered Parametric Insurance

for Decentralized Finance

Ibrahim Arogundade

Taiwo Omoyeni

BackstopLabs

August 2026

Version 1.0

Abstract

Backstop is a decentralized insurance protocol for DeFi, settled in an ERC-20 *premium token* (selected by governance at deployment) on Creditcoin. It covers users of any supported protocol against defined loss events: buy coverage, and if a covered event occurs within the policy window, you are paid out.

The mechanism is *proof-triggered and parametric*. Every payout follows from cryptographic proof of a source-chain event that has already occurred. No *live* oracle decides whether a claim is valid or what the payout ratio is; prices enter only as conversions at the event's historical timestamp. There is no claims adjuster and no prediction of future states—only evidence of what already, irreversibly happened.

Pricing is actuarial: a Merton-style model converts a position's risk score into a trigger probability, and an Asset-Specific Risk Factor (ASRF) reserve, borrowed from Basel III bank capital regulation, sizes claims-paying capacity. Neither relies on external price feeds.

Because every trigger is verified against a BlockProver-attested transaction, Backstop removes the latency, manipulation, and trust assumptions that afflict oracle-dependent insurance designs.

Keywords: parametric insurance; decentralized finance; proof-triggered settlement; Merton distance-to-trigger; ASRF reserve; BlockProver attestation

Contents

1 The Problem **5**

1.1 Sudden, Unpreventable Loss Events . . . 5

1.2 Why Existing Solutions Fall Short . . . 5

1.3 The Oracle-Dependency Problem in DeFi Insurance . . . 6

2 Design Philosophy **7**

3 How Backstop Works **8**

3.1 The Scenario . . . 8

3.2 Step 1: Buying Coverage . . . 8

3.3 Step 2: Activation Delay . . . 9

3.4 Step 3: A Liquidation Occurs . . . 9

3.5 Step 4: Claim Submission . . . 9

4 Mathematical Model **11**

4.1 Distance to Default and Trigger Probability . . . 11

4.2 Premium Pricing . . . 11

4.3 Capacity-Based Premium Loading . . . 12

4.4 Loss-Ratio Payout Model . . . 12

4.4.1 Severity Ratio . . . 12

4.4.2 Coverage Ratio . . . 12

4.4.3 Moral Hazard: Exposure Drift Penalty . . . 13

4.5 Why the Payout Model is Oracle-Free . . . 13

4.6 ASRF Portfolio Reserve . . . 13

4.7 Insurable Interest Cap . . . 14

4.8 Duration Bounds . . . 14

4.9 Optimistic Claims: Deterrence Model and Fraud Loading . . . 15

5 Protocol Adapter Framework **17**

5.1 Source-Chain Adapter (`IProtocolAdapter`) . . . 17
5.2 Creditcoin-Side Decoder (`IAdapterDecoder`) . . . 17
5.3 Adding a New Adapter . . . 18
5.4 Beyond Lending . . . 18

6 Capital Structure **19**
6.1 Two Pools, Two Risk Profiles . . . 19
6.2 Unearned Premium Accounting . . . 19
6.3 Withdrawal Mechanics . . . 19
6.4 Premium Token . . . 20
6.5 Why Stablecoins Should Be the Premium Token . . . 20

7 Attestcoin Protocol Integration **21**
7.1 BlockProver Precompile . . . 21
7.2 What is Proved vs. What is a Trusted Input . . . 21
7.3 Height-to-Time Anchor . . . 21
7.4 Continuity Proof . . . 22

8 Security and Threat Model **23**
8.1 Adversary Assumptions . . . 23
8.2 Threat Table . . . 23
8.3 Disclosed Residual Risks . . . 23

9 Governance **25**
9.1 Governed Parameters . . . 25
9.1.1 Per-Adapter Parameters . . . 25
9.2 Immutable Elements . . . 26
9.3 Adapter Governance . . . 26
9.4 Premium Token Migration . . . 26

10 Roadmap **27**

11 Resources 28

12 Team 29

13 Appendix 30

13.1 Notation Reference 30

13.2 Default Parameter Values 32

13.3 Worked Numerical Scenario: Maria 32

13.4 Worked Numerical Scenario: Large Position 34

13.5 Worked Numerical Scenario: Over-Liquidation (Drift Penalty) 34

1 The Problem

1.1 Sudden, Unpreventable Loss Events

DeFi exposes users to loss events that are sudden, irreversible, and largely outside their control. The canonical example is a liquidation: lending protocols such as Aave and Compound require borrowers to post collateral exceeding the borrowed value, and when the collateral-to-debt ratio falls below a protocol-defined threshold, the position is liquidated—typically at a discount that penalizes the borrower [1]. A borrower holding $100,000 in collateral on Aave V3 with a health factor approaching 1.0 faces a binary outcome: either the position is liquidated (realizing a loss of 5–15% of collateral value) or the borrower must top up collateral before the next block [2]. In volatile markets this margin of safety can evaporate in seconds.

The same shape recurs across DeFi: validator slashing, derivative position closures, and stablecoin de-pegs all convert a sudden, provable on-chain event into a realized loss. Users are left to bear these losses uninsured. In lending, over-collateralization serves as the primary risk buffer, but the cushion is uneven: larger borrowers can absorb slippage; smaller ones cannot. Liquidation penalties function as an implicit insurance premium paid to keepers, yet the borrower has no control over the price at which liquidation executes.

1.2 Why Existing Solutions Fall Short

Centralized insurance. Protocols such as Nexus Mutual and InsurAce pool capital and pay claims after manual or semi-automated assessment. The model works but introduces trust assumptions: a claims committee, oracle feeds, or governance votes determine whether and how much to pay. These trust surfaces are precisely the ones that DeFi was designed to eliminate.

Real-time monitoring designs. A recurring proposal in DeFi is to prevent loss events by monitoring position risk in real time and triggering protective actions (e.g., auto-deleveraging) before the event occurs. This approach has two structural flaws:

1. **Unavoidable latency.** A monitoring bot reads state, computes a response, and submits a transaction. Between the read and the inclusion of the response transaction, the on-chain state may have changed. In a fast-moving market, the protective action arrives *after* the loss threshold has already been crossed. This is not a bug that can be patched; it is a consequence of the asynchronous nature of blockchain execution.
2. **Race-condition asymmetry.** Liquidation bots are economically incentivized to act faster than protective bots. An MEV searcher running the same monitoring logic will always have a structural advantage: they can front-run the protective transaction and capture the arbitrage profit [3]. No amount of additional monitoring infrastructure changes this game-theoretic reality.

Oracle-dependent insurance. Even protocols that style themselves as “decentralized insurance” frequently depend on Chainlink or similar oracle networks [4] to determine whether a claim is valid. This introduces three risks: (i) oracle downtime can freeze the claims process; (ii) oracle manipulation can trigger false claims; (iii) the oracle itself becomes a single point of failure that must be trusted.

1.3 The Oracle-Dependency Problem in DeFi Insurance

The core issue is one of *epistemic posture*: should an insurance protocol act on what it *observes now* (via an oracle) or on what it can *prove happened* (via cryptographic attestation)? The former requires continuous trust in the oracle's availability and correctness. The latter requires only that a transaction was included in a block whose header is attested by the BlockProver—a purely cryptographic primitive with no availability assumption.

2 Design Philosophy

Backstop is governed by a single design rule:

> *Only ever act on proof of something that already, irreversibly happened.*

This rule has two consequences:

1. **No live-state guarantees.** Backstop cannot guarantee that a position will not suffer a loss event tomorrow. It can only guarantee that a position met the risk threshold at the time coverage was activated, and that if a covered loss event *does* occur during the coverage window, the policyholder receives a payout proportional to the realized loss.
2. **No oracle dependency.** Because every decision is made from a BlockProver-attested transaction, Backstop never needs to read live on-chain state from the source protocol. The risk score and exposure are reconstructed from proven logs, and the trigger event is decoded from a proven loss-event log—data that is immutable once attested. Prices enter only to normalize amounts to a common USD basis (see section 6.4); they never decide an outcome.

This design trades *prevention* for *settlement*. Backstop does not try to prevent loss events; it pays the policyholder after a covered loss event occurs. The trade-off is deliberate: prevention requires prediction (which requires oracles), while settlement requires only proof—which requires only cryptography [5].

The result is a protocol that is:

- **Oracle-free settlement**—no live price feed determines whether a claim is valid or what the payout ratio is. All prices enter as *conversions*, never as decisions: the premium-token USD rate (see section 6.4) converts quoted USD amounts into premium-token units (at the current rate when buying, and at the source event's historical rate when settling), and an adapter's per-asset USD feeds normalize proven trigger amounts to the canonical USD base at the event's historical timestamp. Both are sourced from a governance-settable oracle today and a decentralized feed (Chainlink/Pyth) in production, and affect only unit conversion, never claim validity or the payout ratio.
- **Manually unadjustable**—no committee can redirect a payout; the math determines the outcome.
- **Async-safe**—no race conditions, no front-running of protective transactions.
- **Cross-chain by construction**—the BlockProver attests to transactions on any supported source chain; the settlement logic on Creditcoin is chain-agnostic.

3 How Backstop Works

This section walks through a complete policy lifecycle using a concrete example.

3.1 The Scenario

Maria holds a borrow position on Aave V3 (Ethereum mainnet). Her position has:

- Total collateral: $100,000 (in base asset units)
- Current health factor: 1.35 (i.e., 35% above the liquidation threshold)

Maria is concerned about a volatile week ahead. She buys Backstop coverage.

3.2 Step 1: Buying Coverage

Maria calls `buyPolicy` on the Backstop contract (deployed on Creditcoin) with parameters specifying:

- The source chain (Ethereum) and protocol (Aave V3)
- A proven risk snapshot: she provides a BlockProver-attested Aave transaction showing her health factor of 1.35 and collateral of $100,000
- Desired payout maximum: $50,000
- Coverage horizon: 7 days

The protocol:

1. Verifies the risk snapshot proof via the AttestationGateway (BlockProver precompile).
2. Decodes the risk score (health factor = 1.35) and exposure ($100,000) from the proven event.
3. Requires the buyer to be the position owner of the covered subject. No third party, sponsor, or delegated buyer may purchase coverage on someone else's position.
4. Checks the insurable-interest cap: $\$50{,}000 \leq \$100{,}000 \times 1.5 = \$150{,}000$ ✓.
5. Computes the premium using Merton-style pricing (see section 4): $\$50{,}000 \times P_{\text{trigger}} \times 1.20 \times \text{loading}(C) \approx \$239$. The 1.20 multiplier is the *Aave adapter's* underwriting margin—pricing and risk thresholds are per-adapter, not platform-wide (see section 5).
6. Pulls the premium in the premium token, splits it: protocol fee $\rightarrow$ Treasury, net stake $\rightarrow$ claims pool.
7. Mints a policy in `PendingActivation` status.

3.3 Step 2: Activation Delay

Maria's policy enters a mandatory activation delay (e.g., 15 seconds on testnet; configurable per-adapter by governance). This window prevents purchasing coverage *after* a loss event has already occurred but before the proof is available—a critical anti-front-running measure.

After the delay, Maria (or anyone) calls `activatePolicy`, providing a new BlockProver-attested risk snapshot from *after* the purchase. The protocol verifies:

- The risk event timestamp is at or after the purchase time.
- The position owner in the snapshot matches Maria's address.
- The health factor at activation (≥ 1.02, the Aave adapter's minimum risk score—a per-adapter threshold) confirms the position was still solvent.

The policy transitions to `Active` with:

- `riskScore0` = 1.35 (baseline health factor)
- `exposure0` = $100,000 (baseline collateral)
- `expiresAt` = now + 7 days

3.4 Step 3: A Liquidation Occurs

Three days later, a market crash drops Maria's health factor below 1.0. A liquidation bot executes a `LiquidationCall` on Aave, seizing part of her collateral. The realized loss is $8,000.

3.5 Step 4: Claim Submission

Maria (or anyone) calls `submitClaim`, providing a BlockProver-attested proof of the Aave `LiquidationCall` transaction. The protocol:

1. Verifies the proof via the AttestationGateway.
2. Decodes the trigger event: severity = $8,000 (debt liquidated), exposure at event = $8,000 (collateral seized, below the $100,000 baseline).
3. Confirms the event timestamp falls within the coverage window.
4. Computes:

$$\text{severity ratio} = \min\left(\frac{\$8{,}000}{\$100{,}000},\ 1\right) = 0.08 \tag{1}$$

$$\text{exposure drift} = \max\left(\frac{\$8{,}000 - \$100{,}000}{\$100{,}000},\ 0\right) = 0 \ \text{ (collateral seized} \leq \text{baseline, so drift penalty} = 0) \tag{2}$$

$$\text{payout} = \$50{,}000 \times 0.08 \times 0.80 \times (1 - 0) = \$3{,}200 \tag{3}$$

$3,200 is committed against the claims pool for Maria's beneficiary address. Because the Aave adapter enables the optimistic claim manager (`disputeLivenessPeriod` = 2 days), the payout is *not* paid immediately: the policy moves to `ClaimPending`, and the exact payout is released to the beneficiary only after the evidence window closes and the arbitrator returns a `Valid` verdict (see section 4.9). An adapter that disables the claim manager (`disputeLivenessPeriod` = 0) settles the identical math in the same transaction, paying the beneficiary directly.

4 Mathematical Model

All quantities are WAD-scaled (WAD $= 10^{18}$) unless otherwise noted.

4.1 Distance to Default and Trigger Probability

Backstop adapts the Merton structural credit-risk model [6]. The "distance to default" measures how far the risk score is from the trigger threshold (riskScore $= 1$), normalized by volatility and time.

Definition 4.1 (Distance to Trigger) *Given a risk score s (higher = safer), annualized volatility σ, and coverage horizon T (in years):*

$$d_2 = \frac{\ln(s) - \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}} \tag{4}$$

The trigger probability is the standard normal CDF evaluated at $-d_2$:

Definition 4.2 (Trigger Probability)

$$P_{\text{trigger}} = \mathsf{N}(-d_2) = \frac{1}{2}\left[1 + \mathsf{erf}\left(\frac{-d_2}{\sqrt{2}}\right)\right] \tag{5}$$

Remark 4.1 *The CDF is implemented via the Abramowitz & Stegun approximation (error $\leq 1.5 \times 10^{-7}$) [7] in `NormalDist.sol`; the inverse CDF uses the Acklam rational approximation [8]. The natural logarithm uses an atanh power series and the exponential a range-reduced Taylor series, both WAD-scaled, in `ExpMath.sol`.*

4.2 Premium Pricing

The gross premium for a policy is:

Definition 4.3 (Premium)

$$\textit{premium} = \texttt{payoutMax} \times P_{\text{trigger}} \times (1 + m) \times L(C) \tag{6}$$

where:

- m *is the underwriting margin—a* per-adapter *parameter declared on the registry (no platform fallback), e.g., 20%*
- $L(C)$ *is the capacity-loading multiplier at utilization* C

The protocol-wide fraud-loading term (section 4.9) is added to m to give the effective margin used in every quote; it defaults to zero in the current deployment.

4.3 Capacity-Based Premium Loading

The capacity utilization is the ratio of the portfolio reserve requirement to total value locked:

$$C = \frac{R}{\text{TVL}} \tag{7}$$

The loading multiplier is a piecewise-linear function that increases premiums as the pool approaches full utilization:

$$L(C) = \begin{cases} 1 & \text{if } C \le C_{\text{optimal}} \\ 1 + \kappa_1 \cdot \left(C - C_{\text{optimal}}\right) & \text{if } C_{\text{optimal}} < C \le C_{\text{max}} \\ 1 + \kappa_1 \cdot \left(C_{\text{max}} - C_{\text{optimal}}\right) + \kappa_2 \cdot \left(C - C_{\text{max}}\right) & \text{if } C > C_{\text{max}} \end{cases} \tag{8}$$

where κ_1 and κ_2 are governance-set slope parameters (all quantities are WAD-scaled in the contracts; $\kappa_1 = 2.0$ and $\kappa_2 = 20.0$ by default). At the hard cap C_{hardCap}, new policy issuance is blocked entirely.

4.4 Loss-Ratio Payout Model

The payout is a deterministic function of the proven trigger event:

Definition 4.4 (Payout)

$$\textit{payout} = \texttt{payoutMax} \times \underbrace{\min\left(\frac{\textit{severity}}{\texttt{exposure0}},\ 1\right)}_{\textit{severity ratio}} \times \texttt{coverageRatio} \times \underbrace{(1-\delta)}_{\textit{drift adjustment}} \tag{9}$$

4.4.1 Severity Ratio

The severity ratio is the realized loss as a fraction of the baseline exposure. Both severity and exposure are decoded from the proven source-chain event and normalized to the same canonical USD (μUSD) basis—exposure from the adapter's reported `exposureDecimals` via decimal rescaling, and the trigger amounts via the decoder's per-asset USD feeds evaluated at the source event's historical timestamp (see section 6.4). No *live* price feed is needed to settle a claim.

$$s_r = \min\left(\frac{\text{severity}}{\texttt{exposure0}},\ 1\right) \tag{10}$$

4.4.2 Coverage Ratio

The coverage ratio is a *per-adapter covered fraction* (coinsurance) (e.g., 0.80), declared on the registry, so the policy pays 80% of a covered loss and the policyholder retains the remaining 20%—a permanent coinsurance that keeps the policyholder's skin in the game.

4.4.3 Moral Hazard: Exposure Drift Penalty

The trigger decoder reports, alongside severity, the collateral *seized* at the trigger event (`exposureAtEvent`). If the seized value *exceeds* the baseline exposure, the policyholder has effectively let the position grow past the insured baseline (or has been over-liquidated), and the drift penalty reduces the payout accordingly:

$$\delta = \text{clamp}\left(\frac{d - \tau}{\rho_s},\ 0,\ 1\right) \tag{11}$$

where:

- $d = \max\left(\frac{\texttt{exposureAtEvent} - \texttt{exposure0}}{\texttt{exposure0}},\ 0\right)$ is the exposure drift
- τ is the drift tolerance (governance-set, e.g., 10%)
- ρ_s is the drift scaling range (governance-set, e.g., 40%)

4.5 Why the Payout Model is Oracle-Free

The payout formula (eq. (9)) uses only:

1. `payoutMax`—set at purchase, immutable.
2. `severity`—decoded from the proven trigger event (a `LiquidationCall` log) and normalized to USD at the event's historical timestamp.
3. `exposure0`—decoded from the proven activation snapshot.
4. `coverageRatio`—*per-adapter* constant declared on the registry.
5. δ—computed from the proven exposure at event vs. the baseline.

Every input is either a governance constant or is reconstructed from a BlockProver-attested transaction, with prices applied only as USD conversions at the source event's timestamp. There is no point at which Backstop reads *live* on-chain state or a live price feed to determine a payout.

4.6 ASRF Portfolio Reserve

The portfolio reserve requirement uses the **Asset-Specific Risk Factor** (ASRF) model [9] from Basel III internal-ratings-based bank capital regulation [10, 11]. This is not an original contribution but an established framework for correlated default risk.

Definition 4.5 (ASRF Conditional Default Probability) *Given a marginal trigger probability p_i, systemic asset correlation ρ, and confidence level α:*

$$p_i^* = \mathsf{N}\left(\frac{\mathsf{N}^{-1}(p_i) + \sqrt{\rho} \cdot \mathsf{N}^{-1}(\alpha)}{\sqrt{1-\rho}}\right) \tag{12}$$

The reserve contribution of a single policy is:

Definition 4.6 (ASRF Contribution)

$$r_i = \mathit{payoutMax} \times \mathit{coverageRatio} \times p_i^* \tag{13}$$

Remark 4.2 *The trigger probability p_i is frozen at activation time and never changes during the policy's life. This means the portfolio reserve requirement is the* sum *of constant per-policy contributions, maintained in $O(1)$ per policy mutation. There is no need to re-evaluate any risk model as conditions change.*

Remark 4.3 (Probabilistic reserve floor) *The ASRF reserve floor is a* statistical *bound, not a hard guarantee. At confidence level $\alpha = 0.99$, there is a 1% probability that actual losses exceed the reserve. The floor is also capped at the pool's total value locked (TVL) to prevent setting an impossible target. In extreme scenarios where realized losses exceed the reserve, claim payouts are limited to the available pool assets. This is consistent with the Basel III ASRF framework: the reserve sizes the pool for normal stress; catastrophic tail risk is absorbed by the LP capital itself.*

4.7 Insurable Interest Cap

To prevent speculative over-insurance, the payout maximum is capped as a multiple k of the covered exposure:

$$\texttt{payoutMax} \leq k \times \texttt{exposure} \tag{14}$$

This cap is enforced both at purchase and at activation (since exposure may have changed). At purchase, the same cap is additionally enforced *cumulatively per subject*: the policy's `payoutMax` plus the subject's already-active coverage may not exceed $k\times$ the exposure, preventing a user from splitting one position's insurable value across many policies. The factor k (e.g., 1.5) is governance-set and represents the maximum leverage ratio the protocol is willing to underwrite.

4.8 Duration Bounds

Policies have a maximum coverage horizon set *per-adapter* on the registry:

$$0 < \texttt{horizon} \leq \texttt{maxHorizon} \tag{15}$$

Active policies may be extended by paying additional premium for the extension period, subject to the same per-adapter maximum total duration:

$$\texttt{horizon}_{\text{initial}} + \texttt{extension} \leq \texttt{maxHorizon} \tag{16}$$

Remark 4.4 (Extension window) *When a policy is extended, the new expiry is computed as $\mathit{expiresAt}_{new} = \mathit{expiresAt}_{old} + \mathit{extension}$. The coverage window extends from the* current expiry, *not from the time of extension. This prevents a user from resetting a nearly-expired policy to gain a fresh coverage window at a potentially different risk score. The additional premium is re-priced using the policy's baseline risk score and volatility (frozen at activation), so a worsening position cannot be re-rated; the extension is nevertheless subject to the then-current capacity utilization and the per-adapter margin.*

4.9 Optimistic Claims: Deterrence Model and Fraud Loading

Claims settled through the `OptimisticClaimManager` are routed through an evidence-staking middleware. The claim manager holds no tokens: when a policy claim is routed to it, the payout plus a premium-fraction bonus is marked *committed* against the ReservePool—no money leaves the pool and no party is paid before a verdict. The covered policy moves to a `ClaimPending` state for the evidence and decision windows. It leaves that state in one of two ways: the arbitrator resolves the claim (`Claimed` on a Valid verdict, paying the beneficiary the exact escrowed payout; or `ClaimRejected` on a Fraudulent one, paying nothing and returning the escrow to the pool, restoring LP NAV), or—if no evidence was ever submitted—anyone finalizes the genuinely uncontested claim as Valid once the decision window lapses, paying the beneficiary. The evidence window (`disputeLivenessPeriod`) is a per-adapter parameter; the decision window (the arbitrator's gap after evidence closes) is protocol-wide and owner-settable (3 days by default). This subsection describes the evidence model, its economics for evidence providers and liquidity providers (LPs), and the protocol-wide fraud-loading term that prices the residual, undetected-fraud risk into every premium.

Evidence-Staking Model

During a per-adapter evidence window, any actor may submit *arbitrary evidence bytes* for an asserted claim, backed by a stake S_i. Providers do not declare a position; at resolution the arbitrator assesses each provider's evidence and assigns an *unsigned* weight $w_i \in [0, 1]$ (WAD-scaled):

- $w_i > 0$ asserts the evidence supports the claim being *fraudulent*;
- $w_i = 0$ is neutral (the stake is returned unchanged).

There is no minimum or maximum stake (only a nonzero floor) — the more staked, the more exposure to the outcome. Stakes are pulled into the ReservePool and held as *committed* (unearned) until the provider redeems. Let $E_i = w_i \cdot S_i$ be provider i's exposure. At resolution the arbitrator issues a verdict $v \in \{\text{Valid}, \text{Fraudulent}\}$ and the math (in the `EvidenceLib` library) settles each provider:

- **Fraudulent verdict:** every provider with $w_i > 0$ is a winner and recovers their stake plus a share of the reward pool proportional to their exposure E_i, where the reward pool is the escrowed premium bonus:

$$\text{rewardPool} = \texttt{netStake} \cdot \texttt{evidenceRewardFraction} \tag{17}$$

  The bonus is committed against the pool at assertion time, so the reward pool is always fully backed.
- **Valid verdict:** every provider with $w_i > 0$ staked (possibly fake) fraud evidence and is slashed by $\texttt{slashRate} \cdot E_i$; the slashed tokens accrue to the pool (LP benefit), and the escrowed bonus returns to the pool.
- **Neutrals** ($w_i = 0$) recover their stake unchanged on either verdict.

On a `Fraudulent` verdict the payout is never paid to the beneficiary: it returns to the pool (restoring LP NAV), and the escrowed premium bonus $\texttt{netStake} \cdot \texttt{evidenceRewardFraction}$ rewards the providers who flagged the fraud, distributed pro-rata to exposure. On a `Valid` verdict the beneficiary is paid the exact escrowed payout from the pool, the escrowed bonus returns to the pool, and providers who staked fraud evidence are slashed, with the slashed amount accruing to LPs. Because the claim manager is a custody middleware, the money movement is exactly the committed-accounting movement: LP share price excludes committed claim assets until they are released or claimed.

Scenario	Outcome
1. Genuine, no evidence	Resolves Valid after the window; beneficiary paid
2. Genuine, fake fraud evidence	Valid verdict; fake-evidence providers slashed, slashed funds accrue to the poo
3. Fraud, no evidence	Finalizes Valid (uncontested); beneficiary paid
4. Fraud, evidenced & proven	Fraudulent verdict; payout returns to pool; flaggers rewarded
5. Fraud, evidenced but unprovable	Neutral weight evidence returns stakes unchanged

Table 1: Fraud scenarios and their consequences for liquidity providers.

Fraud Scenario Catalog

Fraud Loading in Premiums

The premium model (Section 4, eq. (6)) is extended with a protocol-wide residual-fraud loading. Let `margin` be the per-adapter underwriting margin declared on the registry, and `fraudLoadingFactor` a governed engine parameter equal to the expected undetected-fraud rate times the expected loss-given-fraud that the evidence layer does not recover. The effective margin entering every quote is

$$\texttt{effectiveMargin} = \texttt{margin} + \texttt{fraudLoadingFactor}, \tag{18}$$

so that

$$\texttt{premium} = \texttt{payoutMax} \cdot P_{\text{trigger}} \cdot (1 + \texttt{effectiveMargin}) \cdot \texttt{loading}(C). \tag{19}$$

The fraud-loading factor is governance-set and periodically recalibrated (the same category as the volatility σ): it rises if realized fraud/dispute outcomes run hotter than modeled and falls if the evidence layer proves more effective. LPs are therefore compensated *in aggregate across the whole book* for the residual fraud the optimistic layer cannot fully eliminate, rather than depending on catching every individual case.

Disclosed Residual Risk and Forward-Looking Defenses

The system does not claim fraud is eliminable; the residual risk is disclosed and priced (Section 8.3). A bond posted by the claimant would be a weak deterrent on its own — at realistic detection probabilities the required bond would be several multiples of the payout — so the protocol instead incentivizes *detection* by paying evidence providers a fraction of the policy premium when they are right and slashing them when they are wrong. Further defenses are designed but not yet implemented, in increasing order of complexity:

- **Extended audit window**: a longer (e.g. 90-day) clawback window funded from a reserve set-aside, so funding-graph evidence that does not expire can still surface fraud after the fast liveness window closes.
- **Reputation stake**: a one-time refundable stake to register as a policyholder, slashed on any proven fraud, raising the attacker's total capital at risk across the whole relationship.
- **Repeat-offender loading**: folding any wallet cluster with a proven fraud history into a permanently elevated capacity-loading rate.

All funds—escrow, evidence stakes, and the premium bonus—live in the `ReservePool`; the claim manager only commits and releases them. This keeps a single source of truth for LP net-asset value and ensures no money leaves the pool before a verdict.

5 Protocol Adapter Framework

Backstop is designed to generalize across lending protocols, proof-of-stake chains, derivatives platforms, and other DeFi niches. The generalization is achieved through a two-contract adapter pattern.

5.1 Source-Chain Adapter (`IProtocolAdapter`)

Deployed on the source chain (e.g., Ethereum mainnet), the adapter:

- Reads live on-chain state from the target protocol (e.g., Aave V3's `getUserAccountData`).
- Emits a canonical `RiskSnapshotEvent` containing:
  - `positionOwner`—the address that owns the position
  - `subjectId`—a protocol-specific identifier (for Aave V3: `keccak256(abi.encode(positionOwner))`, which the Creditcoin-side decoder reproduces from the proven log)
  - `riskScore`—the protocol's native risk metric (e.g., health factor), higher = safer
  - `exposure`—the position's USD-denominated value at `exposureDecimals` decimals (Aave V3 reports aUSD base units at 8 decimals; a USDC-denominated source would report 6)
  - `exposureDecimals`—the decimal basis of `exposure`, so the decoder can normalize it to the canonical μUSD base (10^6 μUSD= \$1) by decimal rescaling alone
- Defines a trigger event topic (e.g., the `LiquidationCall` signature for Aave).
- Is bound at construction to a single source protocol contract (`CONTEXT`, e.g., the Aave V3 Pool), so callers cannot redirect it to a fabricated pool.

The adapter interface is intentionally minimal: it knows nothing about Creditcoin, proof verification, or settlement logic.

5.2 Creditcoin-Side Decoder (`IAdapterDecoder`)

Deployed on Creditcoin, the decoder:

- Parses proven source-chain transaction logs into normalized structs (`RiskSnapshot`, `TriggerEvent`).
- Normalizes each adapter's native exposure to the canonical μUSD base using the reported `exposureDecimals` (`PriceLib.normalizeUsd`).
- Decodes the adapter's trigger event (e.g., Aave's `LiquidationCall`), converting the proven raw amounts to μUSD via per-asset oracle feeds evaluated at the source event's historical timestamp.
- Provides the volatility key and trigger event topic for the core contracts.
- Is whitelisted via the `ProtocolAdapterRegistry` through a timelocked propose/execute flow (the delay is a deployment-configurable immutable; the current testnet deployment sets it to zero, so proposals execute immediately).

5.3 Adding a New Adapter

To support a new protocol:

1. Deploy a source-chain adapter that emits `RiskSnapshotEvent`.
2. Deploy a Creditcoin-side decoder that implements `IAdapterDecoder`.
3. Governance proposes the adapter via `proposeAdapter`, specifying the source chain, protocol ID, decoder address, and source-chain adapter address.
4. After the timelock delay (a deployment-configurable immutable, currently zero on testnet), anyone calls `executeProposal` to activate the adapter.
5. Governance declares the `triggerEmitter`—the source-chain contract (e.g., the Aave V3 Pool) whose proven logs count as loss events.
6. Governance sets the mandatory per-adapter configuration (`setAdapterConfig`): activation delay, horizon and payout caps, entry risk score, coverage ratio, margins and fees, policy-count limits, the coverage-window buffer, and the optimistic claim-manager economics. There is no platform fallback—an adapter is not usable until configured.
7. Governance sets the volatility parameter for the adapter's volatility key.

5.4 Beyond Lending

The adapter framework is niche-agnostic. Potential extensions include:

- **Proof-of-stake slashing:** An adapter reads slashing events from a validator contract; the trigger event is the `Slashed` log.
- **Derivatives:** An adapter reads liquidation events from a perpetuals DEX; the risk score is the margin ratio.
- **RWA tokenization:** An adapter reads attestation events from a real-world asset custody contract.

6 Capital Structure

6.1 Two Pools, Two Risk Profiles

Backstop separates capital into two distinct pools that are never merged:

1. **Underwriting Pool (ReservePool):** LPs deposit the premium token and receive LP tokens (ERC-20) representing their pro-rata share. This pool pays claims. LPs earn yield from:
   - Net premiums (gross premium minus protocol fee) that become earned when policies settle—at expiry, claim, cancellation, or void.
   - The share-price appreciation from premium-funded inflows (the `fund` path does not mint new shares).
2. **Treasury:** Receives protocol fees (underwriting fee on premiums, processing fee on refunds). Sweepable by governance.

6.2 Unearned Premium Accounting

When a policy is purchased, the net premium is recorded as *unearned* in the `totalUnearnedPremium` accumulator. Funds committed to open optimistic claims (payout escrow plus evidence stakes) are likewise held in a `committedClaims` accumulator. Both are excluded from the LP share price, so LPs cannot realize them until they are earned or released.

$$\text{share price} = \frac{\text{total assets} - \text{unearned premium} - \text{committed claims}}{\text{LP token supply}} \tag{20}$$

This prevents two abuses:

- **Quick pool boost:** A large premium payment cannot inflate the LP share price, because it is excluded from the withdrawable base until earned.
- **Over-profiting:** LPs cannot withdraw unearned premium as profit. The accumulator is released in full when a policy settles (expiry, claim, cancellation, or void), regardless of when during the term it settles.

6.3 Withdrawal Mechanics

LP redemptions are subject to two gates:

1. **Reserve floor:** Redemptions are blocked if they would take total assets below the ASRF reserve requirement. Claims are senior to exits.
2. **Utilization gate:** Redemptions are blocked when capacity utilization C exceeds the soft cap C_{softCap}.

There is no withdrawal cooldown: redemptions execute immediately once both gates pass. Additionally, the withdrawable base for share pricing always excludes both unearned premium and committed claim assets (escrow and evidence stakes), so LP redemptions can never touch funds backing open claims or unearned policies.

6.4 Premium Token

The protocol is designed to be token-agnostic. The *premium token* is the ERC-20 token chosen at deployment as the unit in which premiums are paid, claims are settled, reserves are held, and LP shares are denominated. All coverage amounts (`payoutMax`, exposure), premiums, and payouts are quoted in *USD* (μUSD, where 10^6 μUSD= \$1); the premium token is the unit of *payment*, not the unit of *account*.

Because coverage is USD-denominated but settlement happens in an arbitrary ERC-20, the protocol converts between the two at the premium token's USD price. That price is read through a single `IPriceOracle` interface and applied by `PriceLib` to every USD↔token conversion—premiums, claim payouts, and the ASRF reserve contribution alike. On testnet the price is supplied by `PriceOracle`, a governance-owned, owner-settable value (a disclosed trusted input, analogous to the height→time anchor). In production the same interface is backed by a decentralized price feed (e.g. Chainlink or Pyth), so the protocol does not depend on any particular price source. Whichever source is wired in, prices enter only as *conversions*, never as decisions: they convert between USD and the premium token and normalize an adapter's trigger amounts to USD at the source event's historical timestamp. They never determine whether a claim is valid or what the payout ratio is.

This design decouples the protocol from any single asset. In the current deployment the premium token is fixed at deployment (e.g. USDC) and changing it requires redeploying the ReservePool and Treasury (see section 9.4); the unitless math means such a change never alters any pricing or reserve formula, only the settlement unit.

All protocol math is *unitless*: the trigger probability, severity ratio, coverage ratio, and ASRF reserve are pure ratios. A claim therefore computes a USD-denominated payout from proven event data; the price-dependent steps are converting the trigger amounts to USD at the source event's historical timestamp and converting the resulting USD payout into premium-token units. The oracle is expected to enforce staleness and sanity bounds so a frozen or absurd price cannot silently corrupt a conversion (section 4.4).

6.5 Why Stablecoins Should Be the Premium Token

While the framework is token-agnostic, governance should strongly prefer a stablecoin such as USDC or USDT as the premium token:

- **Unit stability.** A USD-pegged token keeps the USD-denominated exposure, the quoted premium, and the actually-paid token in the same unit. A volatile premium token would let the real value of coverage drift with its price, undermining the insurable-interest cap and the ASRF reserve sizing.
- **Actuarial accuracy.** The Merton pricing model and the severity ratio assume a stable unit. If the premium token's USD value moves materially over the policy horizon, the effective coverage purchased—and the reserve held—no longer matches the modeled risk.
- **Minimal price sensitivity.** For a USD-pegged token the oracle price is ≈ 1 and effectively constant; a de-pegging event would need to be extreme to move settlement value materially. A volatile token would make every premium, payout, and reserve read sensitive to the oracle price—and to its staleness.
- **Liquidity.** LPs deposit and redeem in a liquid, familiar asset. Deep secondary markets for USDC/USDT make deposits, redemptions, and claim payouts frictionless.
- **Operational maturity.** Widely adopted stablecoins have established issuance, redemption, custody, and audit infrastructure.

For these reasons the protocol's reference deployment uses USDC as the premium token on Creditcoin.

7 Attestcoin Protocol Integration

Backstop is built on Creditcoin and relies on the Attestcoin BlockProver for cross-chain transaction verification [12].

7.1 BlockProver Precompile

The `AttestationGateway` contract wraps the BlockProver precompile at address `0x0FD2` on Creditcoin. It exposes two functions:

- `verifyAndEmit(chainKey, blockHeight, encodedTransaction, merkleProof, continuityProof)`—verifies that the transaction was included in a block at the given height on the source chain, and emits the attested data.
- `calculateTxIndex(merkleProof)`—computes the transaction's index within the block, used for replay protection.

7.2 What is Proved vs. What is a Trusted Input

Data	Source	Trust Assumption
Source-chain transaction	BlockProver Merkle proof	Cryptographic (no trust)
Transaction logs	Decoded from proven receipt	Cryptographic (no trust)
Risk score	Adapter's `RiskSnapshotEvent`	Adapter must be whitelisted
Trigger event	Adapter's trigger log	Adapter must be whitelisted
Trigger amount $\rightarrow$ USD	Decoder's per-asset `IPriceOracle` feed at the event timestamp	Disclosed trusted input
Block height $\rightarrow$ timestamp	Governance-set anchor	Disclosed trusted input
Volatility	Governance-set parameter	Disclosed trusted input
Premium-token USD price	`IPriceOracle`: owner-set `PriceOracle` (testnet), Chainlink/Pyth feed (production)	Disclosed trusted input

Table 2: Trust assumptions by data type.

7.3 Height-to-Time Anchor

The BlockProver proves block inclusion by height, but settlement logic needs wall-clock timestamps (for expiry checks, coverage windows). The anchor provides a linear mapping:

$$\texttt{timestamp} = a_{\text{ts}} + (\texttt{blockHeight} - a_h) \times a_{\text{bt}} \tag{21}$$

where a_h, a_{ts}, and a_{bt} are the anchor's block height, Creditcoin timestamp, and average block time, respectively.

Remark 7.1 *The anchor is a* disclosed trusted input*: governance sets it, and it is assumed correct. A manipulated anchor could shift coverage windows, but cannot fabricate or modify source-chain transactions (which are cryptographically attested). The trust surface is narrow and explicit.*

7.4 Continuity Proof

The continuity proof ensures the proven block is connected to a known chain head, preventing isolated or forked block forgeries. It consists of a lower endpoint digest and a sequence of block header roots forming a Merkle chain back to a trusted anchor.

8 Security and Threat Model

8.1 Adversary Assumptions

- The adversary may submit arbitrary transactions to Backstop.
- The adversary may control source-chain adapters (if whitelisted by governance).
- The adversary may observe all public mempool data.
- The adversary *cannot* forge BlockProver attestations (cryptographic assumption).
- The adversary *cannot* control governance (honest majority assumption).

8.2 Threat Table

ID	Threat	Mechanism	Mitigation	Status
T1	Replay of proven transaction	Same proof submitted twice for different claims	`processedQueries` map; query ID bound to policy, chain, height, txIndex	**Closed**
T2	Event spoofing (fake loss event)	Submit a crafted event to trigger payout	Loss events must be logs emitted by the registry-set `triggerEmitter` (the source-chain contract, e.g., Aave V3 Pool); emitter verified on-chain	**Closed**
T3	Log-ordering manipulation	Assume the first matching log is the relevant one	All matching logs enumerated; subject ID filter applied to each	**Closed**
T4	Pre-purchase front-running	Buy coverage after seeing a loss event in the mempool	Activation delay forces new risk snapshot after purchase; stale events rejected	**Closed**
T5	Risk-score manipulation	Artifice a favorable risk score at activation	Risk score decoded from proven adapter event; adapter governance-whitelisted	**Closed**
T6	Exposure inflation	Inflate exposure after purchase to increase payout	Payout uses baseline exposure (`exposure0`) from activation, not event	**Closed**
T7	Oracle manipulation	Manipulate a price to affect payouts	Prices affect unit conversion only (premium-token and per-asset normalization at the event's historical timestamp), never claim validity or the payout ratio; source is governance-swappable (`IPriceOracle`: owner-set today, Chainlink/Pyth in production)	**Closed**
T8	Flash-loan premium manipulation	Use flash loans to inflate pool TVL and reduce capacity loading	Capacity utilization = reserve / TVL; premium computed at purchase time	**Closed**
T9	Adapter upgrade attack	Replace a whitelisted decoder with a malicious one	Timelocked propose/execute flow (deployment-configurable; zero on the current testnet deployment); immediate removal by owner	**Mitigated**
T10	Governance capture	Malicious governance changes risk parameters	Owner-only setters emit public events for all mutations; no timelock on parameter changes (a single-owner risk, see section 8.3)	**Mitigated**
T11	Anchor manipulation	Set a false height-to-time anchor	Disclosed trusted input; cannot forge source-chain transactions	**Mitigated**
T12	LP share-price inflation	Donate the premium token directly to inflate share price	Minimum locked liquidity (`MINIMUM_LIQUIDITY` = 1e6); share price excludes unearned premium and committed claim assets	**Closed**

Table 3: Full threat table. "Closed" = mechanism fully prevents the attack. "Mitigated" = attack is reduced but residual trust in governance or anchor correctness remains.

8.3 Disclosed Residual Risks

1. **Anchor trust.** A governance-set anchor can be incorrect (honest mistake or malicious). Impact: coverage windows shift by up to the anchor error. Cannot cause incorrect payouts (source transactions are still cryptographically attested).

2. **Volatility staleness.** Governance-set volatility is not updated automatically. If actual volatility diverges significantly from the set value, premiums may under- or over-price risk. Mitigated by governance monitoring and periodic updates.

3. **Premium-token and asset prices.** The price sources can be stale or incorrect—a mis-set owner price on testnet, or a stale or faulty external feed in production. Impact: premiums and payouts convert at a non-market rate until corrected. Mitigated by staleness and sanity-bound checks on the oracle read plus governance monitoring. Prices affect unit conversion only, never claim validity or the payout ratio.

4. **Adapter trust.** A whitelisted adapter could emit misleading risk snapshots (e.g., a compromised Aave deployment). The propose/execute process and public proposal events provide detection latency.

5. **Cross-chain finality.** The BlockProver assumes the source chain's finality guarantee. A deep reorg on the source chain could invalidate a previously proven transaction, potentially reversing a payout. Mitigated by waiting for sufficient confirmations.

6. **Single-owner governance.** The core contract uses `Ownable2Step`, meaning a single address controls all risk parameters, adapter whitelists, and fee sweeps. A compromised or malicious owner can alter pricing, block claims, or drain the treasury. For mainnet deployment, the owner should be a multisig or DAO timelock controller. Neither adapter proposals nor parameter changes currently carry a timelock delay (the adapter registry's delay is a deployment-configurable immutable set to zero in the current testnet deployment).

7. **Pause asymmetry.** When the contract is paused, new policy purchases, activations, claims, *and* LP redemptions are all blocked. This prevents LPs from exiting during an incident while claims remain outstanding.

9 Governance

9.1 Governed Parameters

The following platform-wide parameters are mutable by the owner (no timelock; every mutation emits a public event):

Parameter	Description
`driftTolerance`	Exposure drift tolerance before penalty
`driftScalingRange`	Drift penalty scaling range
`rho`	Systemic asset correlation
`alpha`	Reserve confidence level
`insurableInterestFactor`	Max payout / exposure ratio
`kappa1, kappa2`	Capacity-loading slopes
`cOptimal, cMax, cHardCap`	Capacity thresholds
`cSoftCap`	LP redemption gate threshold
`fraudLoadingFactor`	Protocol-wide residual-fraud loading added to every quote's margin
Volatility (per key)	Risk-score volatility per adapter
Premium-token price	`IPriceOracle` μUSDper whole token (owner-set today; Chainlink/Pyth in production)
Height-to-time anchors	Per source chain
Adapter whitelist	Per (chain, protocol) pair
Arbitrator decision window	Optimistic-claim decision gap (default 3 days)

Table 4: Governance-adjustable platform parameters.

9.1.1 Per-Adapter Parameters

The following are declared *per-adapter* on the `ProtocolAdapterRegistry`—there is deliberately no platform fallback. Each supported protocol sets its own values via `setAdapterConfig` (which requires every field non-zero, so an unconfigured adapter is unusable):

Parameter	Description
`activationDelay`	Seconds between purchase and eligibility
`maxHorizon`	Maximum coverage duration
`maxPayout`	Per-policy payout cap
`minRiskScore`	Minimum risk score for activation
`coverageRatio`	Covered fraction / coinsurance (WAD)
`margin`	Underwriting buffer (WAD)
`protocolFeeRate`	Underwriting fee rate
`processingFeeRate`	Refund processing fee rate
`maxPolicies`	Active-policy cap per adapter
`maxActivePerUser`	Active-policy cap per user per adapter
`coverageWindowBuffer`	Padding on the coverage window at claim time
`disputeLivenessPeriod`	Evidence window; 0 disables the optimistic claim manager
`evidenceRewardFraction`	Premium fraction escrowed as the fraud reward
`evidenceSlashRate`	Fraction of a loser's exposure slashed on a Valid verdict

Table 5: Per-adapter configuration (registry, no platform fallback).

9.2 Immutable Elements

The following are fixed by the deployed contracts and cannot be changed without redeployment:

- LP token contract (the ReservePool itself, an ERC-20) and its total supply mechanics, including the `MINIMUM_LIQUIDITY` lock
- The premium token as bound by the ReservePool and Treasury (`immutable` in both, set at deployment; the provider's pointer is owner-settable but the pool/treasury bindings are not)
- OpenZeppelin library versions

Note that the Backstop, ReservePool, Treasury, pricing engine, oracle, registry, gateway, anchor store, transaction decoder, and claim manager addresses are *not* immutable: they are resolved live from the `ProtocolAddressesProvider` and can be swapped by the owner. The ReservePool, in particular, resolves the Backstop live from the provider, and `reApproveReservePool` exists for when a new pool is wired in.

9.3 Adapter Governance

Adapters are added via a two-step propose/execute process with a deployment-configurable timelock:

1. `proposeAdapter`: owner submits the source chain, protocol ID, decoder address, and source-chain adapter address. The proposal is executable after the registry's `TIMELOCK_DELAY` (an immutable set at deployment; currently zero on testnet).
2. `executeProposal`: after the delay, anyone can activate the proposal.

Once active, an adapter is not usable until governance sets its `triggerEmitter` (the source-chain contract allowed to emit loss events) and its mandatory per-adapter configuration via `setAdapterConfig` (see table 5).

Adapters can be immediately removed by the owner (emergency action).

9.4 Premium Token Migration

Because all protocol math is unitless, changing the premium token does not change any pricing or reserve formula—only the unit in which premiums and payouts are denominated. However, in the current architecture the premium token is *not* swappable in place: the ReservePool and Treasury bind it as an immutable at deployment (see section 6.4), and the Backstop core force-approves only the initial token to the pool. A token change therefore requires redeploying the ReservePool and Treasury (and re-approving the new token), rather than an in-place migration. The provider's premium-token pointer is owner-settable, but swapping it alone would desync the pool and treasury from the settlement currency.

In the current deployment the premium token is fixed at deployment (e.g. USDC) and no in-place migration mechanism is implemented.

10 Roadmap

1. **Testnet Phase (Current).** Core contracts deployed on Creditcoin Testnet, with the Aave V3 adapter and the MockLending, MockHackEvent, and MockCarInsurance demo adapters on Sepolia.
   - Deployed addresses on Creditcoin Testnet (chain ID 102031):
     - Backstop: `0x3701dbbcec38b86181b00a55c9d15560a480aa92`
     - PricingEngine: `0x395cb86676974d2525ee07f3c9a2104c19ac4d13`
     - ReservePool: `0x35bf4a35cc84f6d9b4feedcc95c790dc46cab32d`
     - Treasury: `0x9d52d29ecc9bc8962526ea619b9f115579fbf5ec`
     - Premium token (MockERC20): `0xdcf29d5a725192ebae951af13651cf789062d818`
     - ProtocolAdapterRegistry: `0xceaef71ca5f19def325374fca304ca39b3672df3`
     - AttestationGateway: `0x908a1736b1ba53b983261a6b1f9880cf7b3c2fba`
     - AnchorStore: `0xc3d0f358df7a671f264b461630b9fefeadfd9eb3`
     - OptimisticClaimManager: `0x5ea7806a099e6412cb4e4833b0c86895e6f8380c`
     - ProtocolAddressesProvider: `0x7cd86517683de4dd40fc3d188170abe2918b1d26`
     - PriceOracle: `0xcda2c9e1495cba73362401fb09628e3d5c896430`
     - TransactionDecoder: `0xbced7fc0e0ad47b805f80bca3fccf80056fac0e2`
     - AaveV3Decoder: `0x6db39edff16d861dde26ee17dc5318639f83f8d2`
     - MockLendingDecoder: `0x5b06e09c34ade15614933fa5ec955aac1428a78a`
     - MockHackDecoder: `0xe175dd17255d25fb204cc4e665b3de4234052d29`
     - MockCarInsuranceDecoder: `0x033c7718322d4842f72eaabe2058c3bb00919e09`
   - Libraries on Creditcoin Testnet:
     - CollectorLib: `0x4cd40a99e636d2c08b48305a9da29d187712ed78`
     - PricingLib: `0x858baca13a3a516e31e8fa4d6165e95a25106c1b`
     - PoolLib: `0x6c1044ef1b55a0ebbef7893bef9457a5f1368f28`
     - ClaimLib: `0x4e9ca098de3979e0342f14d1042606962693c2e1`
     - PolicyLib: `0x2eafdc6d4a2c85fe8aefee573a3caee3038e6da2`
   - Deployed addresses on Ethereum Sepolia (chain ID 11155111):
     - AaveV3Adapter: `0xe08b55a8ac525ba2a5ab44e362c497eb188d4a57`
     - Aave V3 Pool: `0x6Ae43d3271ff6888e7Fc43Fd7321a503ff738951`
     - MockLending: `0x39b7777d2beec0b2f02784dd41638ac0f81c6f62`
     - MockLendingAdapter: `0x3eb1e8b46aedc63d424adb0038421118410195e8`
     - MockHackEvent: `0xceeef42043464aeac5ae80d438bed790e58bd494`
     - MockHackAdapter: `0xfc15b16c5f0faf3299bb564ccfe4a58719287a71`
     - MockCarInsurance: `0xa7eeec081c1378b3fdcb7c5d488d577e66d3792b`
     - MockCarInsuranceAdapter: `0xa16a60fcedcb8950194438113b7960812a5c838b`
2. **Audit.** Engage independent security audit of core contracts, pricing library, and proof verification flow.
3. **Production price oracle.** Replace the owner-settable `PriceOracle` with a Chainlink or Pyth-backed `IPriceOracle` adapter, with staleness and sanity-bound guards, for the premium-token USD price.
4. **Mainnet.** Deploy to Creditcoin mainnet with Ethereum mainnet Aave V3 adapter.
5. **Additional Protocol Adapters.** Compound V3, Morpho, Euler, MakerDAO vaults.
6. **Additional Chains.** Support for Arbitrum, Optimism, and other EVM chains via additional source-chain adapters.
7. **CEIP Fast-Track.** Pursue Creditcoin Ecosystem Investment Pool fast-track for initial TVL seeding.

11 Resources

Live deployments and operational endpoints:

Resource	URL
Mock adapter demo (Sepolia)	https://backstop-sepolia-mocks.vercel.app/
Backstop dApp (Creditcoin testnet)	https://backstop-cc3.vercel.app/
Hosted subgraph (GraphQL)	https://graph.railbeam.xyz/subgraphs/name/backstop
Expiration worker health endpoint	https://expiration-worker.railbeam.xyz/health

Table 6: Live resources.

12 Team

The BackstopLabs combines deep smart-contract engineering with statistical risk modeling, focused on building trust-minimized insurance infrastructure for DeFi.

- **Ibrahim Arogundade** — Blockchain Engineer. Smart-contract architecture, protocol design, and the Adapter framework.
- **Taiwo Omoyeni** — Statistics and ML. Actuarial pricing model, ASRF reserve sizing, and risk parameter calibration.

13 Appendix

13.1 Notation Reference

Symbol	Definition
P	Premium token (USD-pegged ERC-20, selected at deployment)
μUSD	USD accounting unit (10^6 μUSD= \$1)
p_{USD}	Premium-token USD price (μUSDper whole token, via `IPriceOracle`)
WAD	Fixed-point unit (10^{18})
s	Risk score (higher = safer)
σ	Risk-score volatility (annualized)
T	Coverage horizon (years)
d_2	Distance to trigger
P_{trigger}	Trigger probability
m	Underwriting margin
$L(C)$	Capacity-loading multiplier
C	Capacity utilization (R/TVL)
s_r	Severity ratio
δ	Exposure drift penalty
τ	Drift tolerance
ρ_s	Drift scaling range
ρ	Systemic asset correlation
α	Reserve confidence level
p_i^*	ASRF conditional default probability
r_i	Per-policy ASRF reserve contribution
k	Insurable interest factor
`payoutMax`	Maximum payout (USD, μUSD)
`coverageRatio`	Covered fraction (coinsurance)
`exposure0`	Baseline exposure at activation

Table 7: Full notation reference.

Parameter	Value	Description
τ	0.10	Drift tolerance
ρ_s	0.40	Drift scaling range
ρ	0.20	Systemic correlation
α	0.99	Reserve confidence
k	1.5	Insurable interest factor
κ_1	2.0	Mild loading slope
κ_2	20.0	Steep loading slope
C_{optimal}	0.70	Loading begins
C_{max}	0.90	Steep loading begins
C_{hardCap}	0.95	Issuance blocked
C_{softCap}	0.85	LP redemption gated
`fraudLoadingFactor`	0	Residual-fraud loading on top of the adapter margin
Decision window	3 days	Arbitrator decision gap after the evidence window
σ (Aave V3)	0.80	Aave V3 annualized volatility (mock protocols use 0.75)

Table 8: Default platform parameter values (governance-adjustable).

Parameter	Value	Description
m	0.20	Underwriting margin
`coverageRatio`	0.80	Covered fraction (20% coinsurance)
`minRiskScore`	1.02	Entry guard
`activationDelay`	15 s	Anti-front-running window
`maxHorizon`	365 days	Maximum coverage duration
`maxPayout`	10M μUSD	Per-policy payout cap
`protocolFeeRate`	2%	Underwriting fee on premiums
`processingFeeRate`	0.5%	Refund processing fee
`maxPolicies`	10,000	Active-policy cap per adapter
`maxActivePerUser`	10,000	Active-policy cap per user
`coverageWindowBuffer`	3,600 s	Coverage-window padding at claim time
`disputeLivenessPeriod`	2 days	Evidence window (enables the optimistic claim manager)
`evidenceRewardFraction`	5%	Premium fraction escrowed as the fraud reward
`evidenceSlashRate`	10%	Loser exposure fraction slashed on a Valid verdict

Table 9: Default per-adapter values (Aave V3 reference adapter; each adapter declares its own).

13.2 Default Parameter Values

Platform (PricingEngine / PriceOracle / OptimisticClaimManager)

Per-Adapter (ProtocolAdapterRegistry, no platform fallback)

13.3 Worked Numerical Scenario: Maria

Setup

Position owner (Maria)	`0xCAFE`
Protocol	Aave V3 (Ethereum)
Source chain	Ethereum mainnet (chain key = 1)
Premium token	USDC (default; USD-pegged)
Oracle price	$p_{\text{USD}} = 10^6$ μUSDper USDC (i.e. $1 = 1 USDC)
Collateral at purchase	$100,000 (base units)
Health factor at purchase	1.35
Health factor at activation	1.35
Coverage horizon	7 days ($T = 7/365 \approx 0.01918$ years)
Payout maximum	$50,000
Protocol volatility	$\sigma = 0.80$

Step 1: Premium Calculation

Distance to trigger:

$$d_2 = \frac{\ln(1.35) - \frac{1}{2}(0.80)^2(0.01918)}{0.80\sqrt{0.01918}} \tag{22}$$

$$= \frac{0.3001 - 0.00614}{0.1108} = \frac{0.2940}{0.1108} \approx 2.653 \tag{23}$$

Trigger probability:

$$P_{\text{trigger}} = \mathsf{N}(-2.653) \approx 0.00398 \quad (0.398\%) \tag{24}$$

Base premium (before loading):

$$\text{base} = \$50{,}000 \times 0.00398 \times 1.20 = \$238.80 \tag{25}$$

Capacity loading (assume $C = 0.50$, below $C_{\text{optimal}} = 0.70$):

$$L(0.50) = 1.0 \tag{26}$$

Final premium: $\$238.80$.

Protocol fee (2%, the Aave adapter's `protocolFeeRate`): $\$4.78 \rightarrow$ Treasury.
Net stake (98%): $\$234.02 \rightarrow$ Claims pool.

Step 2: ASRF Reserve Contribution

Marginal trigger probability: $p_i = 0.00398$.

Systemic correlation: $\rho = 0.20$.
Confidence: $\alpha = 0.99$.

Quantiles:

$$\mathsf{N}^{-1}(p_i) = \mathsf{N}^{-1}(0.00398) \approx -2.653 \tag{27}$$

$$\mathsf{N}^{-1}(\alpha) = \mathsf{N}^{-1}(0.99) \approx 2.326 \tag{28}$$

Conditional probability:

$$p_i^* = \mathsf{N}\left(\frac{-2.653 + \sqrt{0.20} \times 2.326}{\sqrt{0.80}}\right) \tag{29}$$

$$= \mathsf{N}\left(\frac{-2.653 + 1.041}{0.8944}\right) \tag{30}$$

$$= \mathsf{N}(-1.803) \approx 0.0357 \tag{31}$$

Reserve contribution:

$$r_i = \$50{,}000 \times 0.80 \times 0.0357 = \$1{,}428.00 \tag{32}$$

This amount is added to the portfolio reserve floor.

Step 3: Claim Settlement

Three days later, a liquidation occurs: severity = $8,000 (debt liquidated), exposure at event = $8,000 (collateral seized).

Severity ratio:

$$s_r = \min\left(\frac{\$8{,}000}{\$100{,}000}, 1\right) = 0.08 \tag{33}$$

Exposure drift:

$$d = \max\left(\frac{\$8{,}000 - \$100{,}000}{\$100{,}000},\ 0\right) = 0 \quad \Rightarrow \quad \delta = 0 \tag{34}$$

Payout:

$$\text{payout} = \$50{,}000 \times 0.08 \times 0.80 \times (1 - 0) = \$3{,}200 \tag{35}$$

$3,200 is committed for Maria's beneficiary. As in the walkthrough (section 3), the Aave adapter's `disputeLivenessPerio` routing applies: the payout is escrowed against the pool and released to the beneficiary after the arbitrator returns a `Valid` verdict; only an adapter with `disputeLivenessPeriod` = 0 pays it immediately.

13.4 Worked Numerical Scenario: Large Position

Setup

A institutional borrower with $500,000 collateral on Aave V3, health factor 1.10, purchasing $400,000 in coverage for 30 days.

Premium

$$d_2 = \frac{\ln(1.10) - \frac{1}{2}(0.80)^2(30/365)}{0.80\sqrt{30/365}} \tag{36}$$

$$= \frac{0.0953 - 0.0263}{0.2294} = \frac{0.0690}{0.2294} \approx 0.301 \tag{37}$$

$$P_{\text{trigger}} = \mathsf{N}(-0.301) \approx 0.3818 \quad (38.18\%) \tag{38}$$

$$\text{base} = \$400{,}000 \times 0.3818 \times 1.20 = \$183{,}264 \tag{39}$$

At $C = 0.50$, $L(C) = 1.0$. Premium = $183,264.

This scenario illustrates why higher risk (lower health factor) and longer horizons produce dramatically higher premiums—the Merton model [6] correctly prices the increased likelihood of a trigger event.

13.5 Worked Numerical Scenario: Over-Liquidation (Drift Penalty)

Setup

A borrower with $100,000 baseline collateral on Aave V3 purchases $50,000 of coverage at a coverage ratio of 0.80. After activation the position is expanded; when a market crash hits, the liquidation seizes more collateral than the baseline ever covered—the moral-hazard case the drift penalty exists for.

Settlement

Severity ratio:

$$s_r = \min\left(\frac{\$60{,}000}{\$100{,}000}, 1\right) = 0.60 \tag{40}$$

Exposure drift (seized value exceeds the covered baseline):

$$d = \max\left(\frac{\$130{,}000 - \$100{,}000}{\$100{,}000}, 0\right) = 0.30 \tag{41}$$

Drift penalty (tolerance $\tau = 0.10$, scaling range $\rho_s = 0.40$):

$$\delta = \text{clamp}\left(\frac{0.30 - 0.10}{0.40},\ 0,\ 1\right) = 0.50 \tag{42}$$

Payout:

$$\text{payout} = \$50{,}000 \times 0.60 \times 0.80 \times (1 - 0.50) = \$12{,}000 \tag{43}$$

Without the penalty the payout would be \$50,000 $\times$ 0.60 $\times$ 0.80 = \$24,000. The moral-hazard reduction halves it: the policyholder let the exposure at the trigger exceed the insured baseline, so the payout is scaled back proportionally.

References

[1] D. Perez, J. Werner, B. Livshits, and S. Arias-Ferreira. Liquidations: DeFi on a knife-edge. In *Financial Cryptography and Data Security (FC)*, 2021.

[2] Aave. Aave V3 documentation: The health factor. https://docs.aave.com/.

[3] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[4] S. Ellis, A. Juels, and S. Nazarov. ChainLink: A decentralized oracle network, 2017. https://research.chain.link/whitepaper-v1.pdf.

[5] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[6] R. C. Merton. On the pricing of corporate debt: The risk structure of interest rates. *The Journal of Finance*, 29(2):449–470, 1974.

[7] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. National Bureau of Standards, Applied Mathematics Series 55, 1972.

[8] P. J. Acklam. An algorithm for computing the inverse normal cumulative distribution function, 2003. https://web.archive.org/web/20151030215612/http://home.online.no/~pjacklam/notes/invnorm/.

[9] O. Vasicek. Loan portfolio value. *Risk*, 15(12):160–162, 2002.

[10] Basel Committee on Banking Supervision. An explanatory note on the Basel II IRB risk weight functions. Technical report, Bank for International Settlements, 2005.

[11] Basel Committee on Banking Supervision. Basel III: A global regulatory framework for more resilient banks and banking systems. Technical report, Bank for International Settlements, 2011.

[12] Gluwa. Creditcoin documentation: Attestation and the Block Prover. https://docs.creditcoin.org/.