SBET V2 / WHITEPAPER

SBET V2 Protocol

A decentralized sports-betting and prediction-market protocol with confidence-adaptive finalization, VRF-selected grader panels, and bond-escalated dispute resolution.

Version: 2.0.0-draft

Date: 2026-04-05

Status: Pre-deployment; Q2 2026 audit scheduled

Target chains: Base, Arbitrum (EVM L2)

Solidity: ^0.8.24 (paris EVM, via-IR, optimizer 200)

License: MIT

Audience: security auditors, protocol integrators, technical investors conducting due diligence on the V2 contract set.

Contract inventory

Contract	Lines	Purpose
SBETCoreV2	923	Match registry, bet accounting, state machine
SBETTreasuryV2	435	Custody, payouts, auto-claim, slash distributor
DisputeManager	424	Bond-escalation dispute state machine
GraderRegistryV2	359	Grader staking, grade submission, slashing
GuardianCouncil	313	Three-tier pause controller
SBETStakerRewards	293	Pull-based MasterChef reward accumulator
ChallengeWindowLib + SignedPositionLib	230	Pure math libraries

Executive Summary

SBET V2 is a ground-up redesign of the SBET sports-betting and prediction-market protocol, targeting EVM L2 deployment on Base and Arbitrum. V2 replaces the V1 single-grader finalization path with a VRF-selected panel of 13 AI graders, a two-phase dispute state machine with bond escalation, and a three-tier emergency pause architecture.

The protocol's security model is economic. A colluding quorum of 9 graders must stake $225,000 collectively — exactly matching the $225k per-match payout cap — so that any successful attack costs the attacker at least as much as it extracts. The attack surface is further bounded by confidence-adaptive challenge windows (90 seconds when graders are unanimous with extreme AI confidence, scaling to 4 hours when dissent appears) and by a dispute bond ladder that forces escalating $5k-$50k capital commitments at each round.

Seven contracts (~130 kB source) implement the protocol under strict CEI discipline with OpenZeppelin's ReentrancyGuard, SafeERC20, AccessControl, and Pausable primitives. Custody lives in a single contract (*SBETTreasuryV2*); the match state machine (*SBETCoreV2*) holds no user funds and cannot release payouts outside of role-gated paths. This document is the engineering source of truth for V2 — pair it with the contracts reference for per-function NatSpec.

13 locked architectural decisions shape V2. Every contract file carries `[Decision #N]` and `[Bug Fix #N]` markers tying source back to the decision log. Four ship-blocking bugs were caught during the V2 skeleton review and fixed before audit. See §4 (Bug Fixes) and §11 (Parameter Appendix).

Key differentiators

Dimension	V2 approach	Why it matters
Finalization latency	90s typical (Fast tier), 15 min / 2h / 4h otherwise	Winners get paid in seconds on uncontested outcomes; contentious outcomes get human review time
Oracle model	VRF-selected 13-grader panel, 9-of-13 quorum, $25k stake each	Attackers cannot pre-select which matches they grade; 4-dropout tolerance
Dispute resolution	Bond escalation up to 15× initial, 5-of-9 arbiter as final court	Griefers pay exponentially per round of delay; arbiter workload throttled
Custody	Treasury is sole custodian; Core holds no user funds	Role-gate on Treasury survives Core compromise
Emergency response	Three-tier pause: match (bond) → category (3-of-5) → global (5-of-9)	Scope-appropriate responses; permissionless match locks prevent single-referee bottlenecks

1. Protocol Overview

V2's seven contracts compose into a single data-flow graph with no cycles. *SBETCoreV2* is the match state machine; *SBETTreasuryV2* is the sole custodian of user stakes; *GraderRegistryV2* holds the grader stakes and grade submissions; *DisputeManager* runs the bond-escalation state machine; *GuardianCouncil* is the three-tier pause controller; and *SBETStakerRewards* is a pull-based MasterChef-style reward accumulator for SBET stakers.

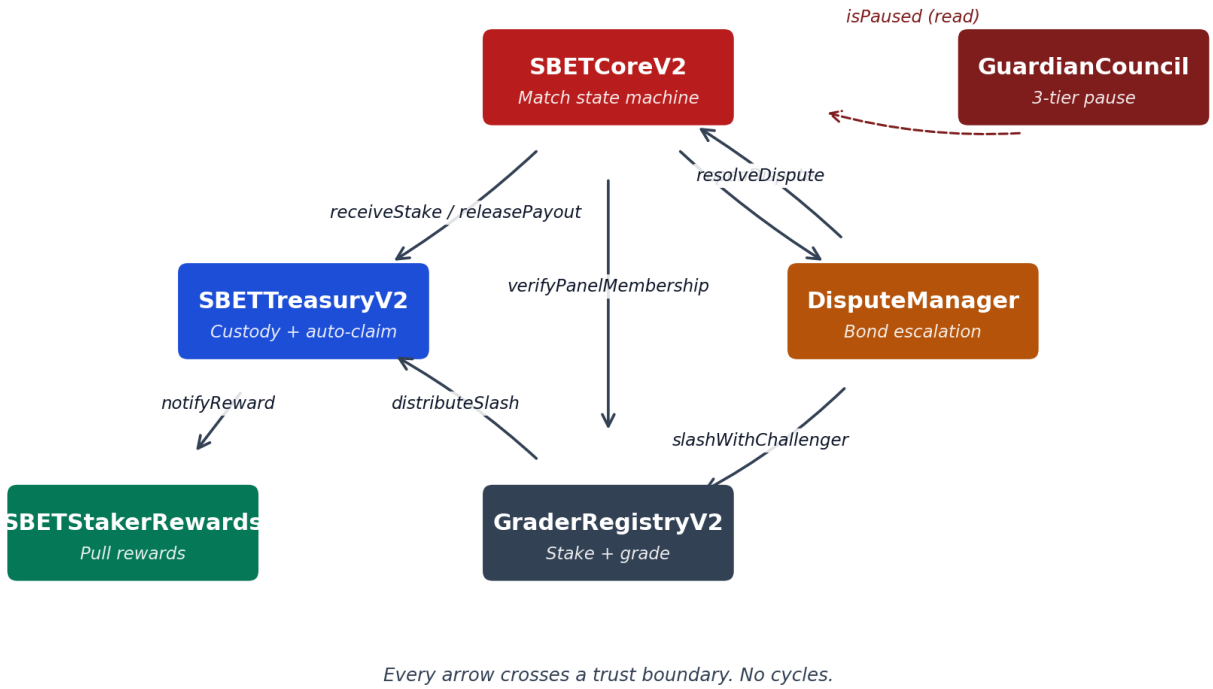


Figure 1. Contract data-flow map. Every arrow crosses a trust boundary. No cycles; Core holds no custody.

1.1 Contract responsibilities

Contract	Key entrypoints	Custody
SBETCoreV2	registerMatch, placeBet, openSignedPosition, proposeOutcome, finalize, openDispute, resolveDispute, claim*, overrideOutcome, lockMatch	None — forwards to Treasury
SBETTreasuryV2	receiveStake, releasePayout, refundVoid, autoClaim (ERC-2771), distributeSlash	User stakes
GraderRegistryV2	register, increaseStake, submitGrade, aggregateConfidence, slashWithChallenger, requestDeregister/executeDeregister	Grader stakes
DisputeManager	openDispute, respond, arbiterResolve, claimByDefault	Dispute bonds
GuardianCouncil	pauseGlobal / unpauseGlobal (5-of-9 + 72h), pauseCategory / unpauseCategory (3-of-5), isPaused views	None
SBETStakerRewards	stake, unstake, harvest, notifyReward (from Treasury)	Staker deposits

1.2 Design principles

- **Non-custodial where possible.** User stakes live in Treasury; Core never holds custody.
- **Fail-closed.** Every write is guarded by explicit match-state, pool-type, and pause checks. No generic `_transition()` hook.
- **Immutable per-match parameters.** `poolType`, `panelRoot`, `panelSize`, and `quorumM` are set once at registerMatch and never mutated.
- **Defense in depth.** OpenZeppelin ReentrancyGuard, SafeERC20, AccessControl, Pausable + CEI discipline on every external function.
- **Separation of authority.** Graders attest; arbiter rules; guardian pauses; governance rotates keys. No single role finalizes unilaterally outside the 5-of-9 arbiter override.

2. Match State Machine

Every match moves through six states: `None` → `Open` → `Proposed` → `{Finalized | Disputed | Voided}`. State is stored as a single byte in `MatchInfo.state`. *PoolType* is **not** a state — it is an immutable attribute set at registration.

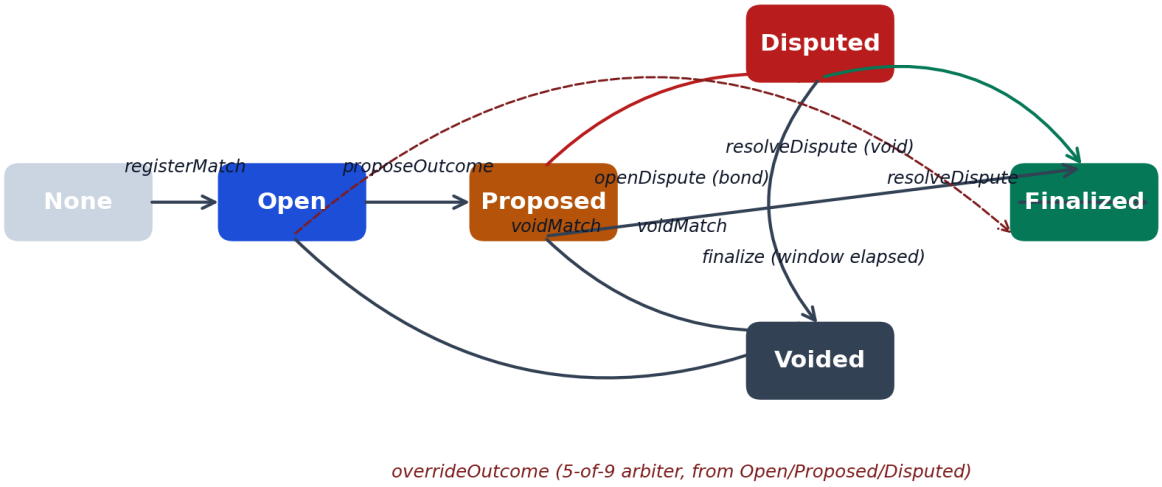


Figure 2. Match state machine. The dashed arc is the 5-of-9 arbiter override, callable from Open, Proposed, or Disputed.

2.1 Transition table (abridged)

From → To	Function	Actor / Role	Key guards
None → Open	registerMatch	MATCH_REGISTRAR_ROLE	panelRoot≠0, quorumM∈(0,panelSize], SignedPosition⇒binary
Open → Proposed	proposeOutcome	GRADER_AGGREGATOR_ROLE	state==Open, signers≥quorumM, not paused
Proposed → Finalized	finalize	permissionless	block.timestamp≥unlockAt, not paused
Proposed → Disputed	openDispute	permissionless + bond	block.timestamp < unlockAt, counterOutcome ≠ outcome
Disputed → {Final,Void}	resolveDispute	DisputeManager only	msg.sender==DISPUTES, state==Disputed
* → Finalized	overrideOutcome	ARBITER_ROLE (5-of-9)	state∈{Open,Proposed,Disputed}
{Open,Proposed} → Voided	voidMatch	GOVERNANCE_ROLE	reason≠DISPUTE

2.2 Invariants

- **Monotonic state.** No external function moves state backwards in the graph.
- **Immutable pool type.** `MatchInfo.poolType` is written at registerMatch and never modified.
- **Immutable panel commitment.** `panelRoot`, `panelSize`, `quorumM` are written at registerMatch and never modified.
- **Cap monotonicity.** `totalAccepted` only increases until finalization.
- **Claim idempotency.** Each claim path zeros the caller's stake before transferring (reentrancy defence).
- **Dispute exclusivity.** A match has at most one open dispute (`_matchDispute[matchId] != 0` blocks re-opening).

3. Economic Security Model

V2's security is economic. Attackers spend money to attack; honest parties earn money to defend; every dishonest outcome is bounded. This section walks through each economic guard with the math behind it, then runs concrete attack scenarios.

3.1 Per-match payout cap

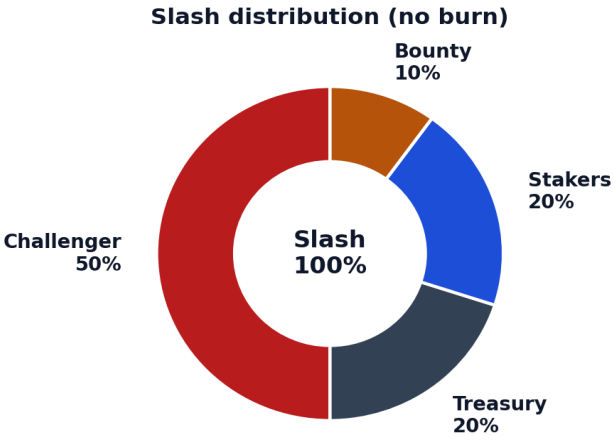
```
uint256 public constant ABSOLUTE_CAP = 225_000e18; // $225k

function perMatchPayoutCap() public view returns (uint256) {
uint256 floor_ = GRADERS.graderStakeFloor(); // default 25_000e18
uint256 candidate = floor_ * _quorumM; // 25k × 9 = 225k
return candidate < ABSOLUTE_CAP ? candidate : ABSOLUTE_CAP;
}
```

The cap is **tied to grader collateral**: `stakeFloor × quorumM = $25k × 9 = $225k`. A fully-colluding quorum loses more money than it can steal. Enforcement is pool-type-specific:

• **PARIMUTUEL** — cap bounds `totalAccepted` (full pot). **[Bug Fix #4]**
• **SIGNED_POSITION** — cap bounds `max(longPool, shortPool)` (long and short offset at settlement; only the larger side is uncollateralized exposure). **[Bug Fix #4]**

3.2 Slashing economics

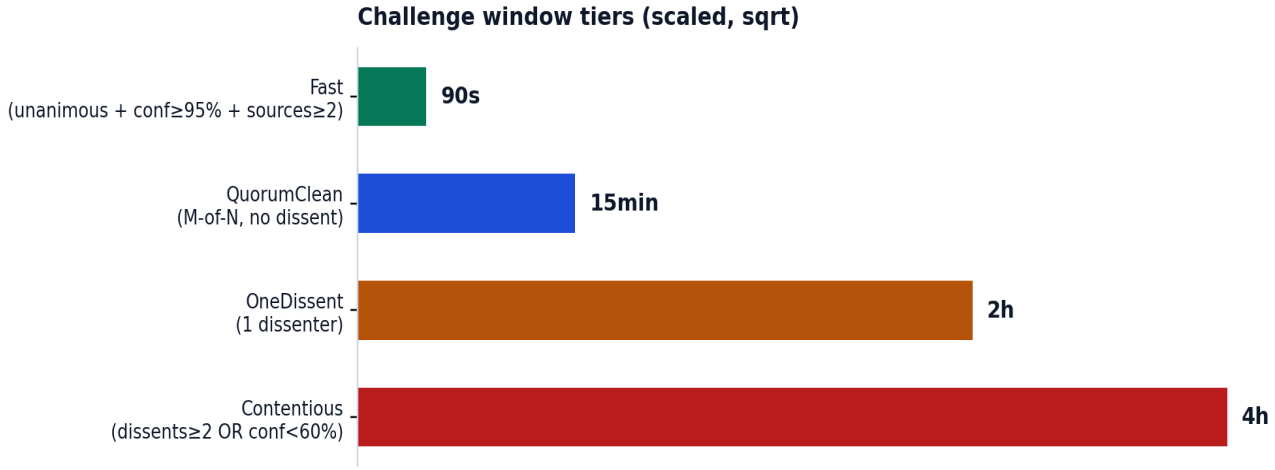


When *DisputeManager* calls `slashWithChallenger`, the slashed amount flows to Treasury and is distributed 50/20/20/10 — no burn. Invariant asserted at Treasury construction: `SLASH_TO_CHALLENGER_BPS + SLASH_TO_TREASURY_BPS + SLASH_TO_STAKERS_BPS + SLASH_TO_BOUNTY_BPS == 10_000`.

The 50% challenger share is the economic reward that makes dispute-raising positive-EV for honest actors: the challenger fronts the bond, carries the operational cost of monitoring, and bears the risk of losing the dispute.

3.3 Dynamic challenge windows

The challenge window is the time between `proposeOutcome` and `finalize`, during which anyone can open a dispute. Window length scales with **consensus quality**:



Tier	Duration	Condition
Fast	90 seconds	unanimous + confBps≥9500 + sourceAgreement≥2/3
QuorumClean	15 minutes	M-of-N met, zero dissent, not fast-eligible
OneDissent	2 hours	exactly 1 dissenting grader
Contentious	4 hours	dissents≥2 OR confBps<6000

Economic reasoning: a rational attacker must open a dispute (≥$5k bond) before `unlockAt` regardless of tier. **Shortening the window does not reduce dispute effectiveness** — it only reduces the delay for the 95%+ of matches that are uncontested.

3.4 Dispute bond escalation

```
initialBond = max(5_000e18, min(50_000e18, matchTvl * 100 / 10_000));
// 1% of match TVL, clamped to [$5k, $50k]
```

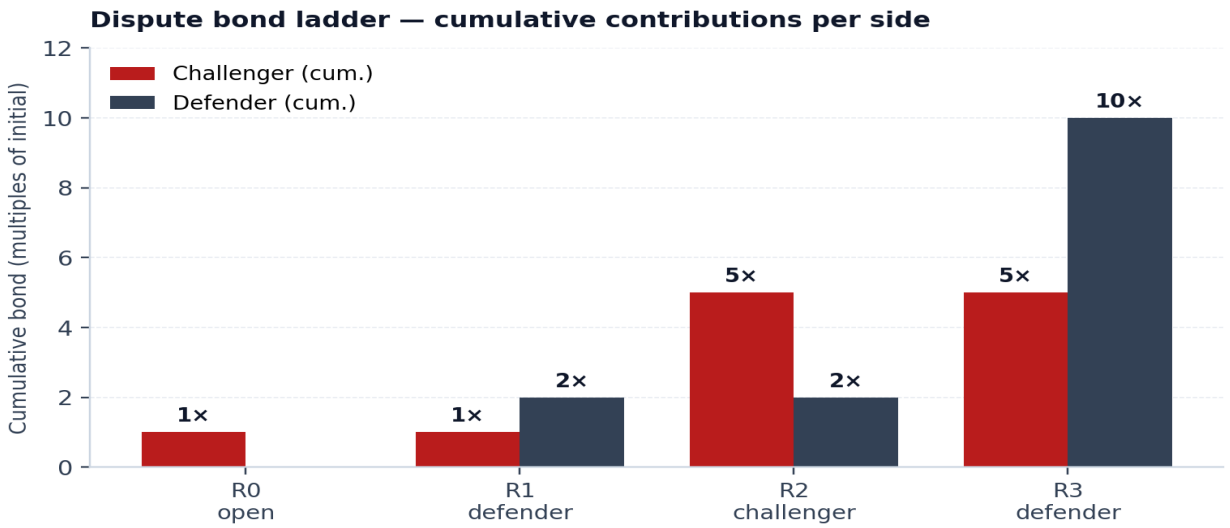


Figure 3. Bond ladder — cumulative contributions per side across 4 rounds. Ceiling cumulative 15× (defender) at round 3.

Response windows are 24h per round. Missing a deadline triggers `claimByDefault`: defaulter loses everything, winner takes all. At max escalation with a $50k initial bond, **$750k of bond capital is locked** in the dispute.

4. Worked Attack Scenarios

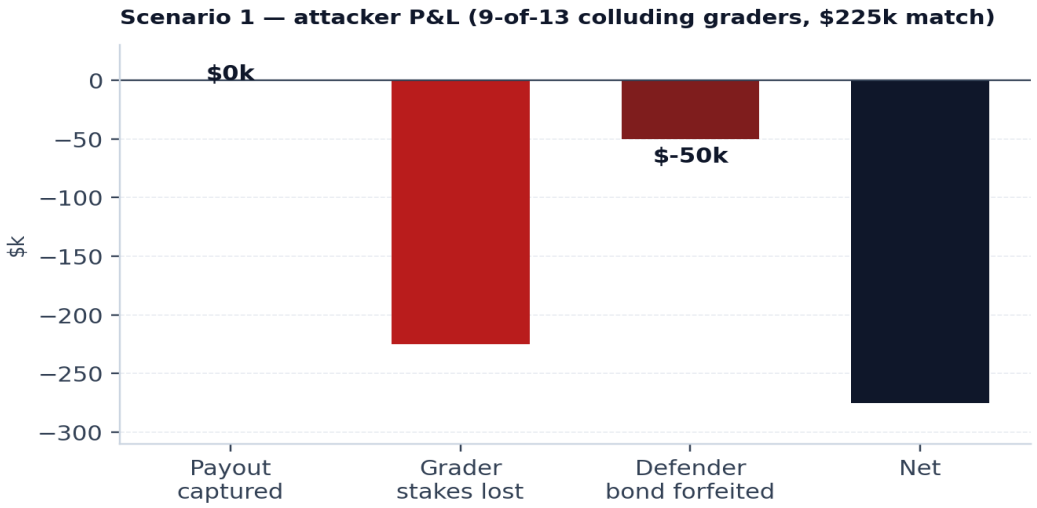
Scenario 1 — Attacker controls 9-of-13 graders

Setup. Attacker has compromised 9 graders (full quorum), each staking $25k SBET → $225k locked. A match with TVL = $225k is registered.

Attacker play. 9 compromised graders sign the wrong outcome. `proposeOutcome` fires with signers=9, dissents=0, confidenceBps=10000, sourceAgreement=3. Challenge window = **Fast tier = 90 seconds**.

Honest response. Within 90s, anyone with ≥$5k dispute bond calls `openDispute(matchId, correctOutcome)`. Bonds escalate. At round 3, state → AwaitingArbiter with $75k total locked.

Arbiter ruling. 5-of-9 Safe multisig reviews evidence, rules for challenger. Attacker's 9 grader stakes (9 × $25k = $225k) are slashed. 50% ($112.5k) flows to challenger. Defender bonds ($50k) also flow to challenger.



Invariant preserved: attacker max loss (~$275k) exceeds max gain (capped at $225k by ABSOLUTE_CAP).

Scenario 2 — Challenge-window race

Attacker wants the 90s Fast-tier to elapse before any dispute is opened. A $5k hot-wallet monitoring bot can call `openDispute` in <10s. To defeat this, the attacker must DoS every monitoring bot network-wide for 90 seconds — infeasible against public, open-source monitoring. Even on successful censorship, the 5-of-9 arbiter can `overrideOutcome` on state {Open, Proposed, Disputed}.

Scenario 3 — Griefer delays finalization

Griefer wants to delay claims on a $10M TVL match. Initial bond = max(5k, min(50k, 10M × 1%)) = $50k (ceiling). Griefer opens, pays $50k. Defender posts $100k. Griefer must post $200k at round 2 — they default. `claimByDefault` → griefer's $50k forfeited, resolution propagates at defender's outcome. **Griefer cost per delay: $50k / 24h, amortized 0.5% of $10M TVL.**

Scenario 4 — Malicious match-lock spam

Attacker locks 100 matches at $1k SBET each = $100k capital. If governance rules the locks malicious, `releaseMatchLock(matchId, true)` slashes each bond to Treasury bounty (10% of slash distribution). Attacker loses $100k; protocol gains $100k revenue. Only upside for attacker: 24h delay per match (if governance does not slash).

Scenario 5 — VRF fallback miner collusion

If VRF does not fulfill within 24h, `seedFromBlockhashFallback` seeds with `blockhash(matchCreationBlock)`. A miner controlling that blockhash within 256 blocks could pre-compute a preferred panel. Mitigation: matches with `VRFSeedFulfilled.fromFallback=true` are visible on-chain; risk engines can flag them; the 4h contentious tier applies if any dissent appears; Scenario 1's economic bound still holds. Accepted as a *liveness guarantee*, not a security guarantee.

5. Grader Oracle

Each grader in V2 runs a **progressive 6-tier AI escalation pipeline**. Grading is not a single model call — it climbs in cost and confidence until one tier yields a result the grader is willing to stake $25k on.

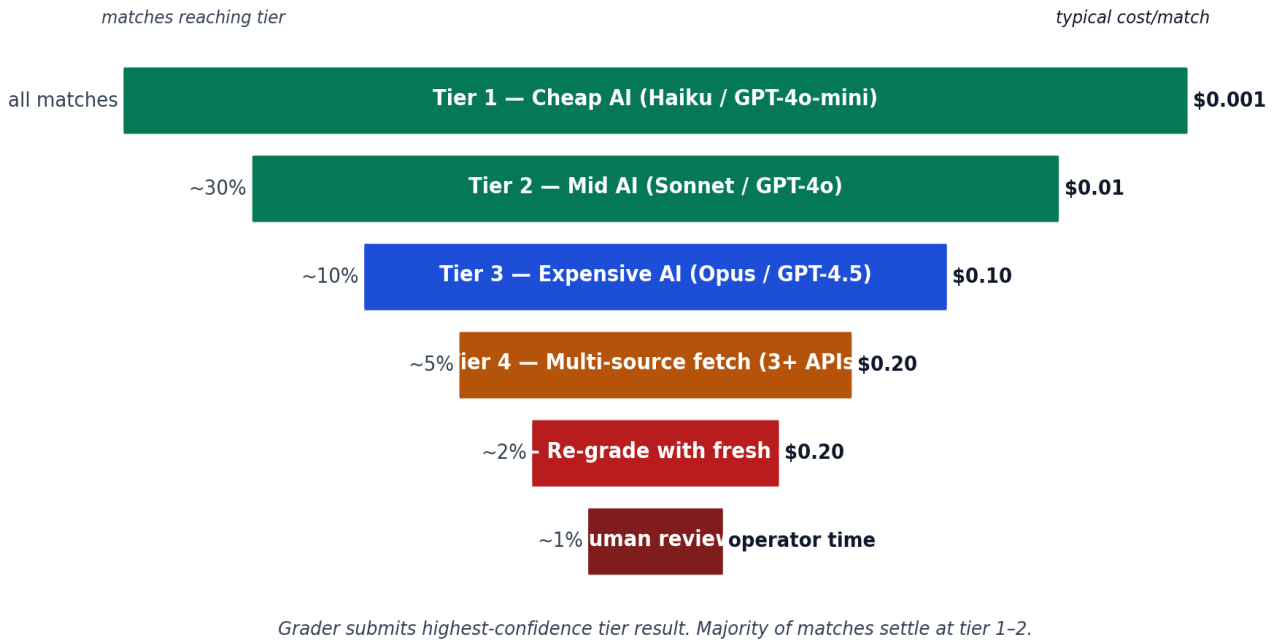


Figure 4. 6-tier escalation funnel with typical per-match cost and fraction of matches reaching each tier.

5.1 Confidence thresholds

Threshold	Value	Consequence
CONFIDENCE_FAST_BPS	9500 (95%)	Fast-tier eligible (with unanimity + source agree ≥ 2)
CONFIDENCE_LOW_BPS	6000 (60%)	Forces 4h Contentious tier
SOURCE_AGREEMENT_FAST	2	≥2 independent sources must agree for Fast tier

5.2 VRF panel selection

Each match is graded by a VRF-drawn sub-panel of 13 graders (quorum 9). At `registerMatch`, the caller commits immutably:

```
panelRoot // keccak256 Merkle root of sorted panel addresses
panelSize // N in M-of-N (default 13)
panelQuorumM // M in M-of-N (default 9)
```

VRF v2.5 supplies the seed; if unfulfilled after 24h, `seedFromBlockhashFallback(matchId)` is permissionless. Graders supply a Merkle proof when calling `submitGrade`; verification is via

`verifyPanelMembership`.

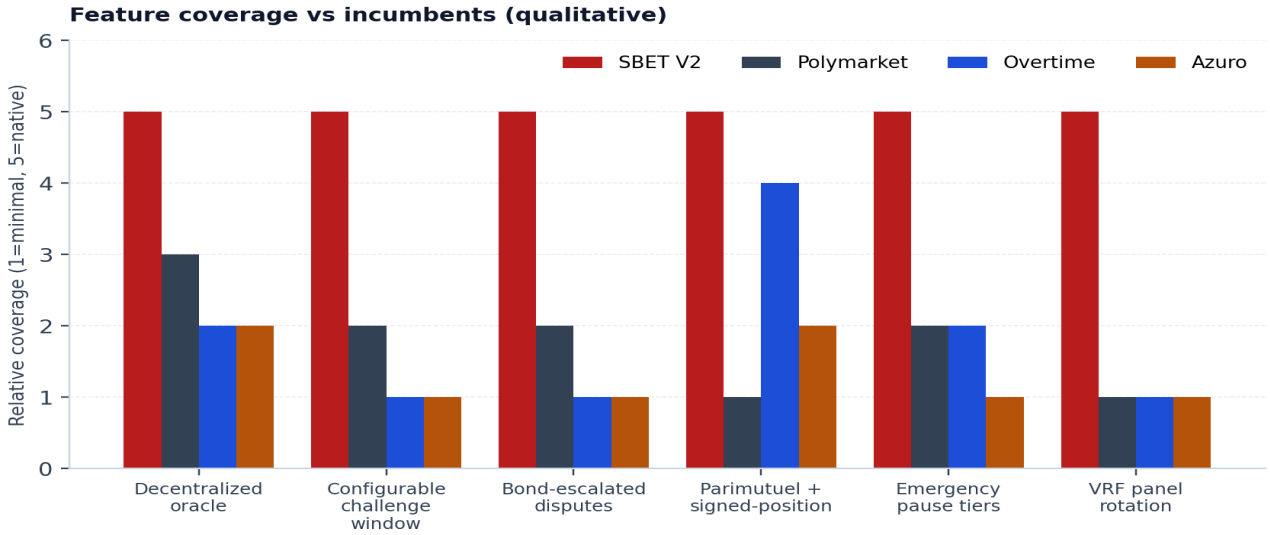
Why randomly-drawn sub-panels: colluders cannot pre-select their matches; failing/offline graders are tolerated (M=9 out of N=13 → 4 dropouts OK); Sybil attacks on specific matches are blocked by the draw.

5.3 Dispute flow (operational view)

1. **R0 (open).** Challenger calls `Core.openDispute(matchId, counterOutcome)`; Core forwards to DisputeManager, which pulls initial bond (1% TVL, clamped [$5k, $50k]).
2. **R1 (defender response).** Defender has 24h to call `respond(disputeId)`; DisputeManager pulls 2× initial.
3. **R2 (challenger response).** Challenger posts 4× initial within 24h.
4. **R3 (defender response).** Defender posts 8× initial. State → AwaitingArbiter.
5. **Arbiter resolution.** 5-of-9 Safe multisig reviews evidence, calls `arbiterResolve(disputeId, winner, finalOutcome, void, evidenceHash)`.
6. **Settlement.** Winner refunded + receives loser's contributions. DisputeManager calls `Core.resolveDispute(matchId, outcome, void, DISPUTE)` — match transitions to Finalized or Voided.

6. Comparison to Incumbents

V2's key distinguishers against the three incumbent decentralized betting / prediction protocols: **Polymarket** (binary prediction markets, UMA optimistic oracle), **Overtime** (AMM-based sportsbook, Thales markets), and **Azuro** (liquidity-pool sportsbook, centralized oracle feeds).



Dimension	SBET V2	Polymarket	Overtime	Azuro
Oracle	13-grader VRF panel + dispute	UMA optimistic	Chainlink feeds	Centralized oracle feeds
Finalization latency	90s / 15m / 2h / 4h (adaptive)	2h+ fixed proposal window	Post-match, fixed	Post-match, fixed
Dispute model	Bond escalation up to 15×, 5-of-9 arbiter	UMA DVM vote (≥2d)	No native dispute layer	No native dispute layer
Pool accounting	Parimutuel + signed-position hybrid	Binary shares (CPMM)	AMM binary/multi	LP vs bettor
Emergency pause	Match / Category / Global (3 tiers)	Admin pause	Admin pause	Admin pause
Per-match cap	$225k, tied to grader collateral	None (pool-bounded)	None (LP-bounded)	None (LP-bounded)

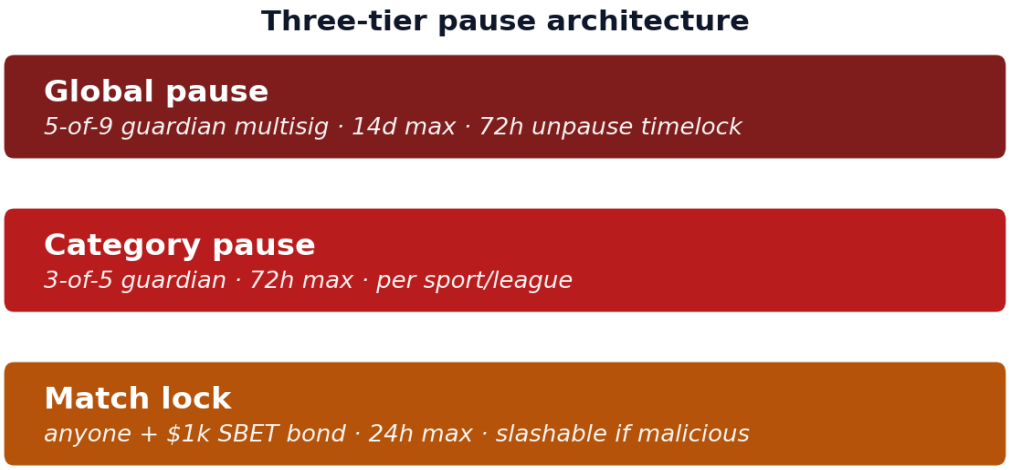
What SBET V2 does differently. No incumbent offers confidence-adaptive finalization: every other protocol pays a flat latency tax for unanimous + high-confidence outcomes. No incumbent ties per-match exposure to oracle collateral (the invariant that a colluding quorum stakes ≥ the max they can extract). SBET's bond-escalation dispute model is closer to Kleros than to UMA's one-shot vote — resolution happens in ≤96h with exponentially growing capital commitments, rather than waiting for a weekly DVM quorum.

7. Custody and Trust Boundaries

Asset	Held by	Released by	Trust assumption
User stakes	SBETTreasuryV2	CORE_ROLE only	Core cannot be spoofed (role-gated)
Grader stakes	GraderRegistryV2	SLASHER_ROLE or grader (after cooldown)	Registry is sole stake custodian
Dispute bonds	DisputeManager	Winner-payout inside DM	Per-dispute bond book
Match-lock bonds	SBETCoreV2	Permissionless on expiry; governance on slash	24h max, slashable on malicious lock
Staker deposits	SBETStakerRewards	Staker unstake (no lockup)	Pull-based rewards

Core holds no user funds. If Core is compromised, user stakes remain protected by the role gate on Treasury. The trusted forwarder in `SBETTreasuryV2` (ERC-2771, for gasless auto-claim) is **immutable** — set in the constructor, never rotated. Forwarder rotation requires a redeploy, by design.

7.1 Pause tiers



Tiers compose. Active tier = highest-scoped active pause.

Pause blocks `placeBet`, `openSignedPosition`, `proposeOutcome`, and `finalize`. It does **not** block `claim*` (winners can always exit on finalize), `claimVoidRefund`, `lockMatch`, or `voidMatch`.

A 72h **unpause timelock** on the Global tier prevents a compromised guardian quorum from pausing-then-unpausing rapidly to mask attacks. Exception: if the Global pause auto-expires (14d elapsed), anyone can unpause with no timelock — the pause was bounded at pause-time.

8. Parameter Appendix

Every configurable parameter, with default value and governance bounds. Source: `SBETCoreV2.sol`, `SBETTreasuryV2.sol`, `GraderRegistryV2.sol`, `DisputeManager.sol`, `GuardianCouncil.sol`, `ChallengeWindowLib.sol`.

8.1 Challenge window

Parameter	Default	Bounds	Where
TIER_FAST_SECONDS	90 s	constant	ChallengeWindowLib
TIER_QUORUM_CLEAN_SECONDS	15 min	constant	ChallengeWindowLib
TIER_ONE_DISSENT_SECONDS	2 h	constant	ChallengeWindowLib
TIER_CONTENTIOUS_SECONDS	4 h	constant	ChallengeWindowLib
CONFIDENCE_FAST_BPS	9500 (95%)	constant	ChallengeWindowLib
CONFIDENCE_LOW_BPS	6000 (60%)	constant	ChallengeWindowLib
SOURCE_AGREEMENT_FAST	2 / 3	constant	ChallengeWindowLib

8.2 Payout cap & slashing

Parameter	Default	Bounds	Where
ABSOLUTE_CAP	$225,000	constant	SBETCoreV2
DEFAULT_STAKE_FLOOR	$25,000 SBET	governance	GraderRegistryV2
SLASH_TO_CHALLENGER_BPS	5000 (50%)	invariant=10000	SBETTreasuryV2
SLASH_TO_TREASURY_BPS	2000 (20%)	invariant=10000	SBETTreasuryV2
SLASH_TO_STAKERS_BPS	2000 (20%)	invariant=10000	SBETTreasuryV2
SLASH_TO_BOUNTY_BPS	1000 (10%)	invariant=10000	SBETTreasuryV2

8.3 Dispute

Parameter	Default	Bounds	Where
INITIAL_BOND_BPS	100 (1% TVL)	governance	DisputeManager
INITIAL_BOND_FLOOR	$5,000	governance	DisputeManager
INITIAL_BOND_CEILING	$50,000	governance	DisputeManager
RESPONSE_WINDOW	24 h	governance	DisputeManager
MAX_ROUNDS	3	constant	DisputeManager
ESCALATION_MULTIPLIER	2×	constant	DisputeManager

8.4 Pause & match-lock

Parameter	Default	Bounds	Where
MATCH_LOCK_BOND	$1,000 SBET	governance	SBETCoreV2
MATCH_LOCK_DURATION	24 h	constant	SBETCoreV2
CATEGORY_PAUSE_MAX	72 h	constant	GuardianCouncil
GLOBAL_PAUSE_MAX	14 d	constant	GuardianCouncil
GLOBAL_UNPAUSE_TIMELOCK	72 h	constant	GuardianCouncil
CATEGORY_QUORUM	3-of-5	governance	GuardianCouncil
GLOBAL_QUORUM	5-of-9	governance	GuardianCouncil

8.5 VRF & panel

Parameter	Default	Bounds	Where
DEFAULT_PANEL_SIZE	13	per-match override	SBETCoreV2
DEFAULT_PANEL_QUORUM	9	(0, panelSize]	SBETCoreV2
VRF_FALLBACK_DELAY	24 h	constant	SBETCoreV2
GRADER_COOLDOWN	7 days	governance	GraderRegistryV2

8.6 Auto-claim (ERC-2771)

Parameter	Default	Bounds	Where
AUTO_CLAIM_FEE_SBET_BPS	200 (2%)	governance	SBETTreasuryV2
AUTO_CLAIM_FEE_OTHER_BPS	350 (3.5%)	governance	SBETTreasuryV2
trustedForwarder	immutable	redeploy only	SBETTreasuryV2 (ERC2771Context)

9. Security Invariants (Formal)

Invariants that every V2 external function must preserve. These are the targets for formal verification and fuzz testing in the Q2 2026 audit.

I1 — Cap invariant

```
∀ match m, m.state ≠ None:
m.poolType == PARIMUTUEL → m.totalAccepted ≤ perMatchPayoutCap()
m.poolType == SIGNED_POSITION → max(m.longPool, m.shortPool) ≤ perMatchPayoutCap()
```

I2 — Collateralization invariant

```
perMatchPayoutCap() ≤ graderStakeFloor × quorumM

// A colluding quorum always stakes at least as much as they can extract.
```

I3 — Slash distribution invariant

```
SLASH_TO_CHALLENGER_BPS + SLASH_TO_TREASURY_BPS +
SLASH_TO_STAKERS_BPS + SLASH_TO_BOUNTY_BPS == 10_000

// Asserted at Treasury construction; no burn.
```

I4 — State monotonicity

No external function may move state backwards in the graph {None, Open, Proposed, Disputed, Finalized, Voided}.

I5 — Immutable pool type

`m.poolType` is written only inside `registerMatch()` and never read-modified-written thereafter.

I6 — Dispute uniqueness

```
_matchDispute[matchId] != 0 ⇒ openDispute(matchId, *) reverts.
```

I7 — Claim idempotency

Each claim path (parimutuel, signed-position, void-refund) zeros the caller's stake accounting *before* transferring, eliminating double-claim via reentrancy. Guarded additionally by OZ `ReentrancyGuard`.

I8 — Panel commitment immutability

`panelRoot`, `panelSize`, `quorumM` are written at `registerMatch` and never modified. The VRF seed may be fulfilled or fallen back, but the *Merkle commitment* is sealed.

Open questions (flagged for audit)

- **On-chain arbiter vote aggregation.** Today ARBITER_ROLE trusts the 5-of-9 Safe off-chain. An on-chain vote phase would make quorum visible on-chain at the cost of more complex state.
- **Dispute pot distribution.** Dispute bonds currently go 100% to the winner. Future revision may route through the 50/20/20/10 split for economic alignment with slash distribution.

- **Grader slashing on arbiter ruling.** DisputeManager.arbiterResolve has a TODO for triggering GraderRegistryV2.slashWithChallenger — requires DM to hold SLASHER_ROLE and a per-match signer-list view.
- **VOID fee-rebate.** claimVoidRefund currently refunds principal only; per-PoolType fee schedule not yet locked.
- **Per-grader key rotation.** Not yet wired; requires governance path distinct from stake lifecycle.

10. Ship-Blocking Bugs Fixed in V2

Four bugs discovered during the V2 redesign cycle. Each is marked `[Bug Fix #N]` in the V2 source.

Bug Fix #1 — Split-outcome parimutuel staking

V1 allowed bettors to stake on multiple outcomes of the same match, breaking parimutuel 'loser pays winner' invariants. V2's placeBet reverts with ExistingPositionDifferentOutcome if a bettor attempts to add stake on a different outcome than their existing position. Constraint is parimutuel-specific; signed-position pools explicitly allow dual long+short per user.

Bug Fix #2 — SBET token reference bootstrapping

V1 looked up the SBET token address dynamically through Treasury, creating a circular dependency at init time and a mid-flight address-change risk. V2 wires SBET_TOKEN as a constructor immutable in both SBETCoreV2 and SBETTreasuryV2. Match-lock bonds use this explicit reference.

Bug Fix #3 — Panel-size source for challenge-window tier

An early V2 draft sourced N (panel size) from the signers count, which would collapse N == M and defeat the dynamic challenge-window tier entirely. Final V2 sources N from MatchInfo.panelSize (committed at registerMatch).

Bug Fix #4 — Per-match cap semantics by pool type

Cap semantics differ between pool types. PARIMUTUEL: cap bounds totalAccepted (full pot). SIGNED_POSITION: cap bounds max(longPool, shortPool) — the two sides offset at settlement, so only the larger represents uncollateralized exposure. Applying a single uniform check would either under- or over-constrain.

License & toolchain

All V2 contracts are MIT-licensed. Compiler settings:

```
[profile.default]
solc_version = "0.8.24"
evm_version = "paris" # L2-portable
via_ir = true
optimizer = true
optimizer_runs = 200
```

OpenZeppelin Contracts v5.x pinned. No custom assembly outside of `unchecked { ++i; }` loop counters. Source repository: `github.com/sbettoken/contracts` (path `v2/`).

Document version: v2.0.0-draft · 2026-04-05 · pre-audit draft. Not for public distribution. Review cycle: security auditors → protocol lead → CTO → external readers (selected investors, integrators).