

IRSB Protocol: IntentScore Calculation

Reputation Metrics with Time Decay

IRSB Protocol Documentation

January 2026

IntentScore Calculation

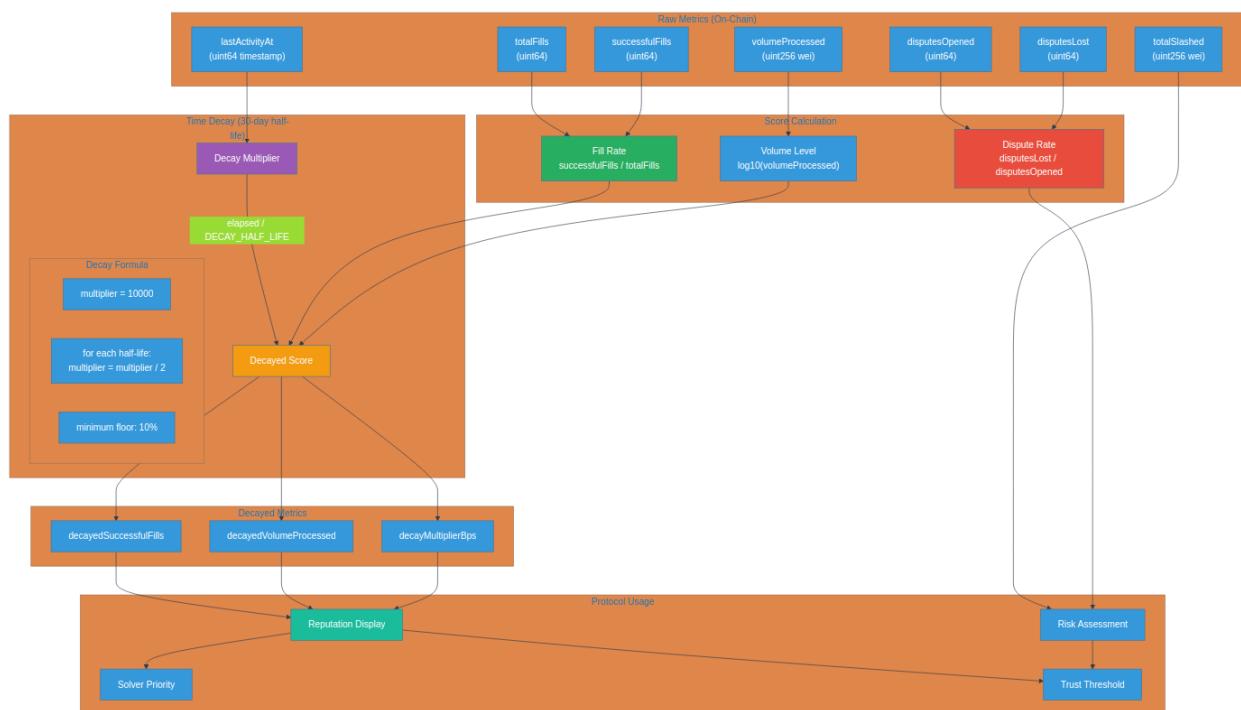


Figure 1: IntentScore Calculation Flowchart

Overview

The IntentScore system tracks solver reputation through on-chain metrics. It provides a transparent, verifiable measure of solver reliability that users and protocols can reference when selecting solvers for intent execution.

Key features:

- **On-Chain Storage:** All metrics stored immutably on Ethereum
 - **Time Decay:** Inactive solvers see reputation naturally decrease
 - **Multi-Dimensional:** Combines fill rate, dispute rate, and volume
 - **Tamper-Resistant:** Only protocol contracts can update scores

Raw Metrics

The IntentScore struct stores six key metrics:

```
struct IntentScore {  
    uint64 totalFills;          // Total receipts posted  
    uint64 successfulFills;    // Finalized without dispute  
    uint64 disputesOpened;    // Disputes against solver  
    uint64 disputesLost;      // Disputes resulting in slash  
    uint256 volumeProcessed;   // Total value processed (wei)  
    uint256 totalSlashed;      // Total amount slashed (wei)  
}
```

Metric Calculations

Fill Rate

The primary success metric:

```
Fill Rate = successfulFills / totalFills
```

- Higher is better
- Tracks percentage of receipts that finalize successfully
- Not affected by dispute outcomes during challenge window

Dispute Rate

Risk indicator:

```
Dispute Rate = disputesLost / disputesOpened
```

- Lower is better
- Measures how often disputes result in slashing
- High rate indicates solver violations

Volume Score

Trust indicator for high-value intents:

```
Volume Score = log10(volumeProcessed)
```

- Logarithmic scaling prevents gaming
 - Solver processing 1 ETH vs 1000 ETH shows different trust levels
 - Useful for filtering solvers for large intents
-

Time Decay Mechanism

The protocol implements reputation decay to ensure inactive solvers don't maintain artificially high scores:

Decay Parameters

```
// 30-day half-life  
uint64 public constant DECAY_HALF_LIFE = 30 days;
```

```

// Minimum floor (10%)
uint16 public constant MIN_DECAY_MULTIPLIER_BPS = 1000;

Decay Formula

function getDecayMultiplier(uint64 lastActivityAt) public view returns (uint16) {
    if (lastActivityAt == 0) return MIN_DECAY_MULTIPLIER_BPS;
    if (block.timestamp <= lastActivityAt) return BPS; // 100%

    uint256 elapsed = block.timestamp - lastActivityAt;
    uint256 halfLives = elapsed / DECAY_HALF_LIFE;

    // After 13+ half-lives, return minimum (10%)
    if (halfLives >= 13) return MIN_DECAY_MULTIPLIER_BPS;

    // Calculate 2^(-halfLives)
    uint256 result = BPS; // Start at 10000 (100%)
    for (uint256 i = 0; i < halfLives; i++) {
        result = result / 2;
    }

    return result < MIN_DECAY_MULTIPLIER_BPS
        ? MIN_DECAY_MULTIPLIER_BPS
        : uint16(result);
}

```

Decay Examples

Days Since Activity	Multiplier	Effective Score
0 days	100%	Full reputation
30 days	50%	Half reputation
60 days	25%	Quarter reputation
90 days	12.5%	Eighth reputation
120+ days	10%	Minimum floor

Reading Decayed Scores

The contract provides a view function for decayed metrics:

```

function getDecayedScore(bytes32 solverId) external view returns (
    uint64 decayedSuccessfulFills,
    uint256 decayedVolumeProcessed,
    uint16 decayMultiplierBps
) {
    Types.Solver storage solver = _solvers[solverId];
    decayMultiplierBps = getDecayMultiplier(solver.lastActivityAt);

    // Apply decay to positive metrics
    decayedSuccessfulFills = uint64(
        (uint256(solver.score.successfulFills) * decayMultiplierBps) / BPS
    );
    decayedVolumeProcessed =

```

```
        (solver.score.volumeProcessed * decayMultiplierBps) / BPS;
    }
```

Note: Negative metrics (disputes lost, total slashed) are NOT decayed - bad history persists.

Score Updates

Scores are updated through authorized contract calls:

On Receipt Finalization

```
function updateScore(
    bytes32 solverId,
    bool success,
    uint256 volume
) external onlyAuthorized {
    solver.score.totalFills++;
    if (success) {
        solver.score.successfulFills++;
    }
    solver.score.volumeProcessed += volume;
    solver.lastActivityAt = uint64(block.timestamp);
}
```

On Dispute Opening

```
function incrementDisputes(bytes32 solverId) external onlyAuthorized {
    _solvers[solverId].score.disputesOpened++;
}
```

On Slashing

```
// During slash() function
solver.score.disputesLost++;
solver.score.totalSlashed += amount;
```

Protocol Usage

User Intent Submission

When submitting an intent, users/protocols can filter solvers by:

1. **Minimum Fill Rate:** Only allow solvers with >95% success rate
2. **Maximum Dispute Rate:** Reject solvers with >5% disputes lost
3. **Minimum Volume:** Require solvers with >100 ETH historical volume
4. **Activity Requirement:** Only active solvers (decayed score > 50%)

Example Filter Query

```
function isQualifiedSolver(bytes32 solverId) external view returns (bool) {
    Types.Solver memory solver = registry.getSolver(solverId);
    Types.IntentScore memory score = solver.score;

    // Active status required
```

```

if (solver.status != Types.SolverStatus.Active) return false;

// Minimum 10 fills
if (score.totalFills < 10) return false;

// Fill rate > 95%
uint256 fillRate = (score.successfulFills * 100) / score.totalFills;
if (fillRate < 95) return false;

// Check decay
(, , uint16 decay) = registry.getDecayedScore(solverId);
if (decay < 5000) return false; // Must have 50%+ reputation

return true;
}

```

Key Concepts

- **lastActivityAt**: Timestamp of most recent receipt, resets decay
 - **Decay Half-Life**: 30 days - reputation halves every month of inactivity
 - **Minimum Floor**: 10% - scores never drop below this threshold
 - **Asymmetric Decay**: Positive metrics decay, negative metrics persist
 - **Volume Logarithm**: Prevents gaming through many small transactions
-

Example Scenario

Solver Reputation Over Time:

Day 1: Solver registers
`score = {fills: 0, disputes: 0, volume: 0}`

Day 1-30: Solver posts 100 receipts, all finalize
`score = {fills: 100, disputes: 0, volume: 50 ETH}`
`decayMultiplier = 100%`

Day 60: Solver inactive for 30 days
`decayMultiplier = 50%`
`decayedSuccessfulFills = 50`
`decayedVolumeProcessed = 25 ETH`

Day 61: Solver posts new receipt
`lastActivityAt = now`
`decayMultiplier resets to 100%`
`fills = 101, disputes = 0, volume = 50.5 ETH`

Day 70: Dispute opened and lost
`disputesOpened = 1`
`disputesLost = 1`
`totalSlashed = 0.1 ETH`
`(These never decay!)`

Review Questions

1. What is the half-life for reputation decay?
 2. Which metrics are affected by time decay?
 3. What is the minimum floor for the decay multiplier?
 4. How is volume score calculated to prevent gaming?
 5. What resets the decay timer for a solver?
-

IRSB Protocol - Transparent, verifiable solver reputation