IRSB Protocol: Receipt Lifecycle

From Intent to Finalization

IRSB Protocol Documentation

January 2026

Receipt Lifecycle

Overview

The receipt lifecycle is the core flow of the IRSB Protocol. It describes how an intent moves from user submission, through solver execution, to final settlement. Understanding this lifecycle is essential for both solvers operating in the ecosystem and users relying on the protocol's guarantees.

The lifecycle consists of three main phases:

1. **Intent Submission** - User submits intent, solver executes
2. **Receipt Posting** - Solver posts cryptographic proof on-chain
3. **Challenge Window** - Time period for disputes before finalization

---

Phase 1: Intent Submission

The intent submission phase happens off-chain:

1. User creates an intent specifying desired outcome (tokens, amounts, chains)
2. Solvers compete to fill the intent
3. Winning solver executes the swap/bridge/transfer
4. Solver prepares receipt with execution details

Intent Structure (Off-chain)

```
struct ConstraintEnvelope {
    uint256[] chainIds;       // Allowed execution chains
    address[] tokensIn;       // Input tokens
    address[] tokensOut;      // Output tokens
    uint256[] minOut;         // Minimum output amounts
    uint256 maxSlippageBps;   // Max slippage in basis points
    uint64 deadline;          // Intent expiration
}
```

Phase 2: Receipt Posting

Once the solver has executed the intent, they post a receipt on-chain:

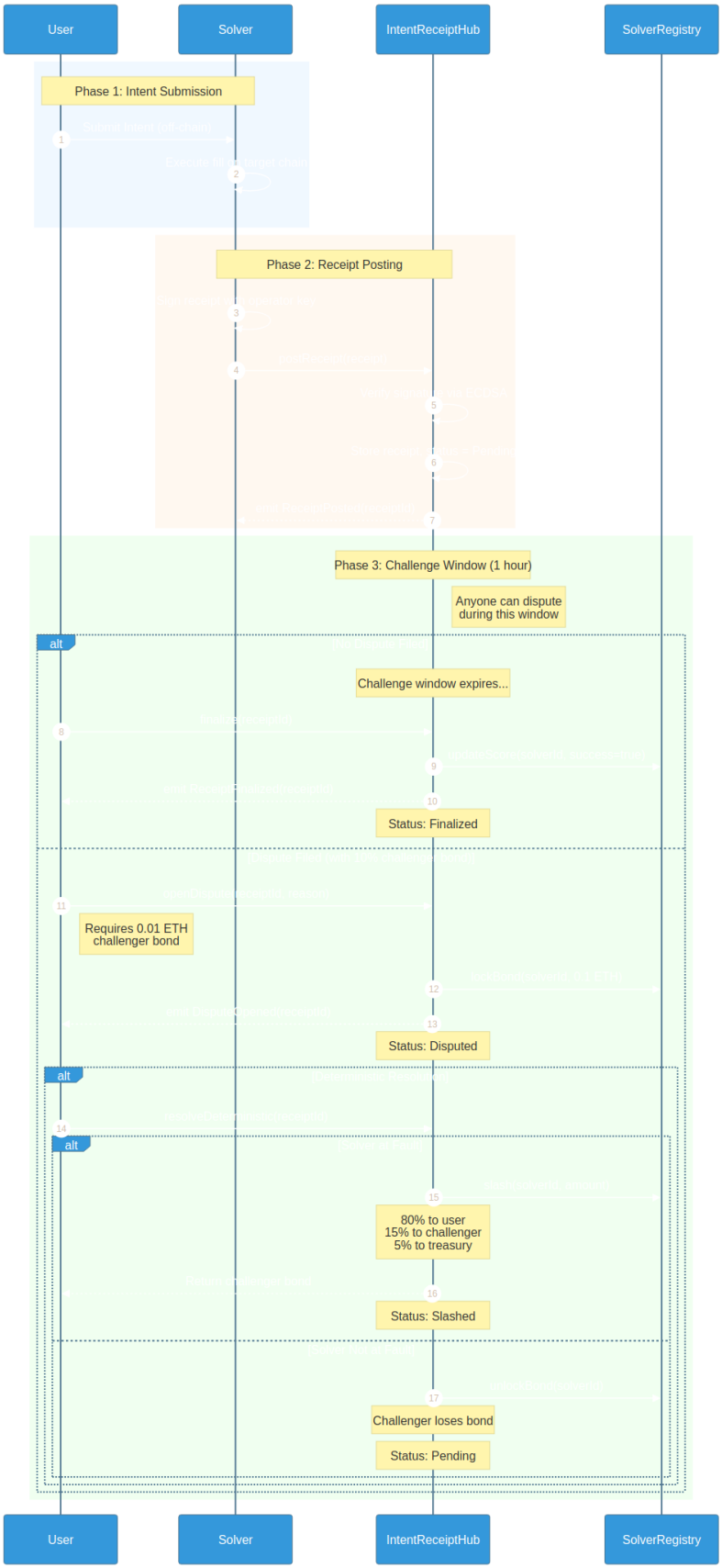

Figure 1: Receipt Lifecycle Sequence Diagram

```
struct IntentReceipt {
    bytes32 intentHash;       // Hash of original intent
    bytes32 constraintsHash;  // Hash of ConstraintEnvelope
    bytes32 routeHash;        // Hash of execution route
    bytes32 outcomeHash;      // Hash of OutcomeEnvelope
    bytes32 evidenceHash;     // IPFS/Arweave CID of proof
    uint64 createdAt;         // Receipt creation timestamp
    uint64 expiry;            // Deadline for settlement proof
    bytes32 solverId;         // Unique solver identifier
    bytes solverSig;          // Solver's signature
}
```

Signature Verification

The protocol uses ECDSA signature verification:

```
bytes32 messageHash = keccak256(abi.encode(
    receipt.intentHash,
    receipt.constraintsHash,
    receipt.routeHash,
    receipt.outcomeHash,
    receipt.evidenceHash,
    receipt.createdAt,
    receipt.expiry,
    receipt.solverId
));
bytes32 ethSignedHash = messageHash.toEthSignedMessageHash();
address signer = ethSignedHash.recover(receipt.solverSig);
```

Phase 3: Challenge Window

The challenge window is a critical security feature:

Parameter	Value	Purpose
Duration	1 hour	Time to detect fraud
Challenger Bond	0.01 ETH	Anti-griefing protection
Solver Bond Lock	0.1 ETH	Collateral during dispute

Opening a Dispute

```
function openDispute(
    bytes32 receiptId,
    Types.DisputeReason reason,
    bytes32 evidenceHash
) external payable;
```

Dispute Reasons:

1. `Timeout` - Expiry passed without settlement
2. `MinOutViolation` - Output below minimum
3. `WrongToken` - Incorrect token delivered
4. `WrongChain` - Settled on wrong chain
5. `WrongRecipient` - Delivered to wrong address

6. `InvalidSignature` - Solver signature invalid
7. `Subjective` - Requires arbitration

Resolution Paths

Path A: No Dispute (Happy Path)

1. Challenge window expires without dispute
2. Anyone can call `finalize(receiptId)`
3. Solver's IntentScore is updated positively
4. Receipt status changes to `Finalized`

Path B: Dispute Filed

1. Challenger posts bond and opens dispute
2. Solver's bond is locked
3. Deterministic resolution attempted:
   - **Solver at fault**: Slashed and distributed
   - **Challenger wrong**: Challenger loses bond

Slashing Distribution

When a solver is found at fault:

Recipient	Percentage	Purpose
User	80%	Compensation for failed intent
Challenger	15%	Reward for catching fraud
Treasury	5%	Protocol sustainability

```
uint16 constant SLASH_USER_BPS = 8000;       // 80%
uint16 constant SLASH_CHALLENGER_BPS = 1500; // 15%
uint16 constant SLASH_TREASURY_BPS = 500;    // 5%
```

Key Concepts

- **Receipt ID**: Deterministic hash of receipt contents for deduplication
- **Challenge Window**: 1-hour period where disputes can be opened
- **Challenger Bond**: 10% of solver bond to prevent griefing
- **Deterministic Resolution**: Automatic verification of timeout/signature
- **Finalization**: Successful completion with no valid disputes

---

Example Scenario

Timeline of a Successful Receipt:

```
T+0:00   Solver posts receipt
T+0:30   User verifies execution was correct
T+1:00   Challenge window expires
T+1:01   Anyone calls finalize()
```

```
T+1:01   Receipt status = Finalized
         Solver IntentScore += 1 successful fill
```

Timeline of a Disputed Receipt:

```
T+0:00   Solver posts receipt
T+0:15   Observer notices MinOutViolation
T+0:16   Observer calls openDispute() with 0.01 ETH
         Solver's 0.1 ETH bond is locked
T+0:20   resolveDeterministic() is called
         Violation confirmed on-chain
         Solver slashed: 0.08 ETH to user
                         0.015 ETH to challenger
                         0.005 ETH to treasury
         Receipt status = Slashed
```

Review Questions

1. What are the three main phases of the receipt lifecycle?
2. What happens if no one disputes a receipt within the challenge window?
3. How much bond must a challenger post to open a dispute?
4. What percentage of slashed funds goes to the user?
5. What fields are included in the signature hash for receipt verification?
