Technical Assessment: UMA Optimistic Oracle Integration

Round 2: Implementation & Depth Evaluation

1. Overview

This assignment evaluates your real-world experience with Solidity and the **UMA Optimistic Oracle (OO v3)**. The task is intentionally small in scope but requires **production-grade quality**, rigorous gas optimization, and comprehensive handling of oracle edge cases.

We are not looking for a tutorial submission. We are testing:

- **Depth of understanding:** How well do you grasp the optimistic assertion pattern?
- **Engineering Rigor:** Attention to security, gas costs, and state management.
- **Edge Case Handling:** How your contract behaves under adversarial conditions.

---

2. Objective

Implement a minimal, gas-optimized Solidity smart contract that:

1. **Accepts Native ETH** from a user to assert a truth (handling the wrapping to WETH internally for UMA bonds).
2. **Submits that assertion** to the UMA Optimistic Oracle (OO v3).
3. **Supports the dispute lifecycle**, allowing users to dispute via your contract (proxying the dispute to UMA).
4. **Handles the UMA resolution callback** securely.
5. **Settles the market** (releases user funds/rewards) only after oracle finality is reached.

---

3. Environment & Constraints

- **Network:** Public Testnet (Sepolia recommended).
- **Solidity Version:** ^0.8.x
- **Oracle:** UMA Optimistic Oracle v3 (Testnet deployment).
- **Currency: ETH** (User facing) -> **WETH** (Oracle facing).
    - *Note: UMA v3 uses ERC20 tokens for bonds. Your contract must handle wrapping ETH to WETH when submitting assertions/disputes.*

---

4. Functional Requirements

4.1 Assertion Creation

The contract must provide a function to:

- Accept ETH from a user (covering the required Bond + optional Market/Bet amount).
- Submit a specific assertion (e.g., *"ETH price was above $2,500 on 1 Feb 2026 (UTC)"*).
- Call assertTruth on the UMA OO v3.
- **Store efficiently:**
    - assertionId
    - Asserter address
    - Assertion timestamp
    - Bond amount
    - Current resolution status

4.2 Dispute Handling (Mandatory)

The contract must allow any address to dispute an assertion within the liveness period.

- The contract should expose a dispute function that accepts ETH (for the dispute bond).
- It must correctly call disputeAssertion on the Oracle.
- **Logic:**
    - Funds and bonds must remain locked during the dispute.
    - **Honest Asserter:** Bond returned to asserter (handled by UMA, verified by your contract).
    - **Dishonest Asserter:** Bond slashed (handled by UMA).
- *Note: Dispute logic must be real and integrated with the Oracle, not mocked.*

4.3 Oracle Callback Handling

Implement the required UMA v3 callback interface (OptimisticOracleV3CallbackRecipientInterface):

Solidity

```
function assertionResolvedCallback(bytes32 assertionId, bool assertedTruthfully) external;
```

- **Security:** Only the UMA Oracle contract may call this function.
- **State:** The callback must execute exactly once per assertion.
- **CEI:** Contract state must be updated *before* any ETH transfers occur.

4.4 Settlement Logic

Settlement must strictly follow UMA's economic guarantees. Funds move **only** after oracle finality.

- **Bond Settlement (Handled by UMA):**
  - Ensure your contract correctly receives the bond back (if it was the payer) or allows the Asserter to claim it.
- **Market Settlement (Handled by You):**
  - **If Assertion == TRUE:** User funds (the "bet") are released to the Asserter.
  - **If Assertion == FALSE:** User funds are released to the Disputer (or counter-party).

Security constraints:

- Prevent early settlement (before finality).
- Prevent double settlement.
- Prevent re-entrancy during ETH transfers (withdrawals).

---

5. Edge Cases (Mandatory)

Your contract must explicitly handle and/or you must document your strategy for:

1. **Last-Second Disputes:** Assertion disputed at the very end of the liveness period.
2. **Concurrency:** Multiple assertions active from the same user simultaneously.
3. **Invalid Callbacks:** assertionResolvedCallback called with an unknown or unrelated assertionId.
4. **Balance Changes:** Assertion resolution occurring after the contract's ETH balance has changed for unrelated reasons.
5. **Re-Submission:** Attempts to resubmit or resettle already resolved assertions.

---

6. Gas Optimization

We expect the code to be optimized for mainnet deployment constraints.

- **Storage:** Minimal storage usage; pack structs where possible.
- **Math:** Use unchecked {} blocks where underflow/overflow is impossible.
- **Variables:** No redundant state variables.
- **Bond:** Use the lowest viable UMA bond amount.
- **Justification:** In your README, explain your choice of Liveness Period and Bond Size.

---

7. Bonus Tasks (High Signal)

Completing these is optional but differentiates "Good" from "Great":

- **Events:** Emit specific lifecycle events (Asserted, Disputed, Resolved).
- **Tests:** Add 1 unit test covering the full Dispute + Resolution flow (using Foundry/Hardhat).
- **Configurability:** Support configurable bond & liveness settings per assertion.
- **Gas Analysis:** Include a comparison of gas costs before vs. after your optimizations.

---

8. Deliverables

1. **Source Code:** Verified Solidity contract(s).
2. **Deployment:** Address of the contract on Sepolia.
3. **README (Max 4 pages):**
    - Explain the UMA Optimistic Oracle flow as implemented.
    - Detail the dispute mechanics.
    - **Crucial:** Explain how you handled the Edge Cases listed in Section 5.
    - Explain your gas optimization decisions.

---

9. Evaluation Criteria

Area	Weight	Description
UMA Oracle Understanding	⭐⭐⭐⭐	Correct implementation of OO v3 flows (assertTruth, assertionResolvedCallback).
Solidity Correctness	⭐⭐⭐⭐	Security, standard patterns, and logic.
Edge Case Handling	⭐⭐⭐⭐	Robustness against invalid states and attacks.
Gas Optimization	⭐⭐⭐	Efficient use of storage and EVM opcodes.
Code Clarity	⭐⭐⭐	Readability and commenting.
Bonus Tasks	⭐⭐	Extra features and testing.

⛔ **Auto-Rejection Conditions:**

- No dispute logic implemented.
- Oracle callback is unprotected (publicly callable by non-oracle).
- Funds moved before oracle finality.
- Generic/Copied README.
- Mocked oracle behavior (not using the actual UMA testnet contracts).

---

10. Time Expectation & References

Time: ~8–9 Hours.

If it takes longer, please focus on Code Correctness over feature completeness and explain in the README.

Reference Material (UMA OO v3):

- UMA OO v3 Documentation
- UMA Testnet Addresses