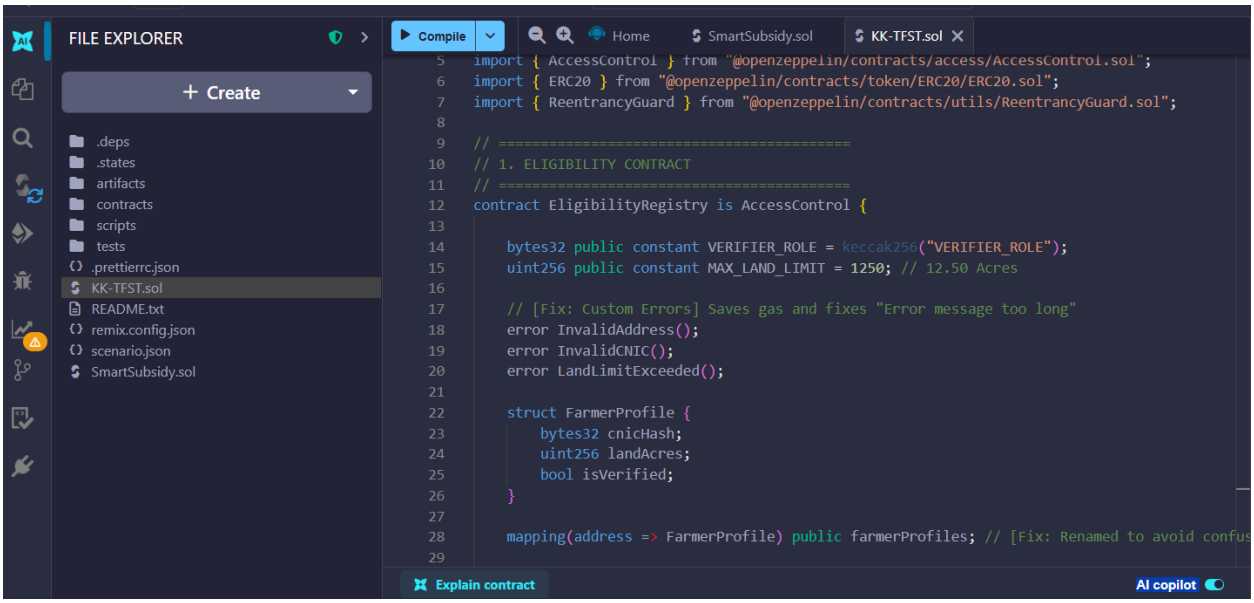
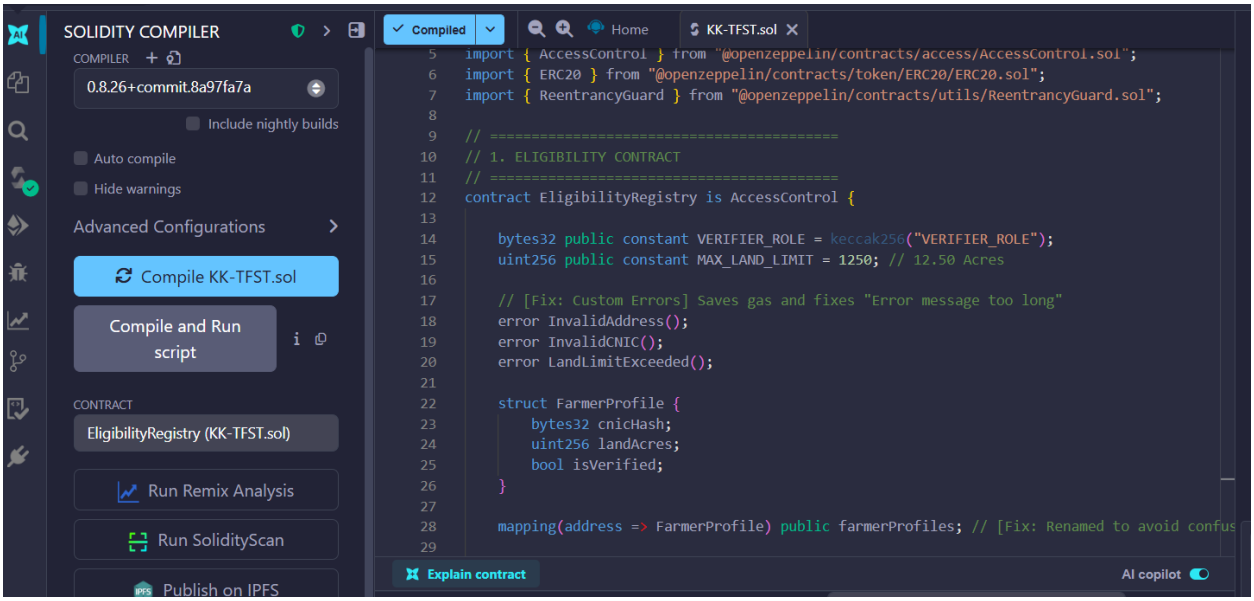
Supplement: Reproducing the TT-TFST Smart Subsidy Contract

Step 1: Setup & Compilation

1. **Create the File:**
    - Open Remix IDE.
    - In the **File Explorer** (left sidebar), create a new file named KK-TFST.sol.
    - Paste the **entire code (available in Annexure)** provided in KK-TFST.sol file into the editor.



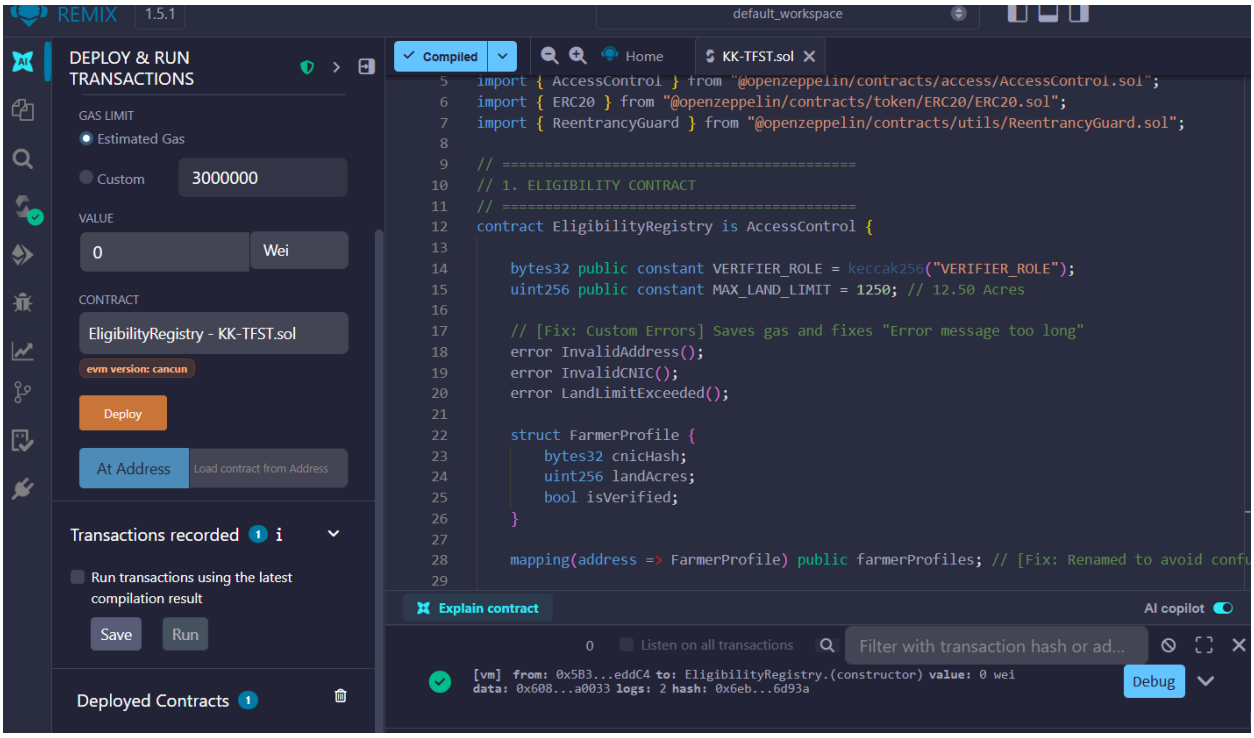
2. **Compile:**
    - Click the **Solidity Compiler** tab (icon looks like an "S").
    - Ensure the **Compiler** version is set to 0.8.26 .
    - Click the blue **Compile KK-TFST.sol** button. Check for the green checkmark indicating success.



Step 2: Deployment (Critical Order)

You must deploy the "database" (Registry) first, then the "logic" (Supply Chain).

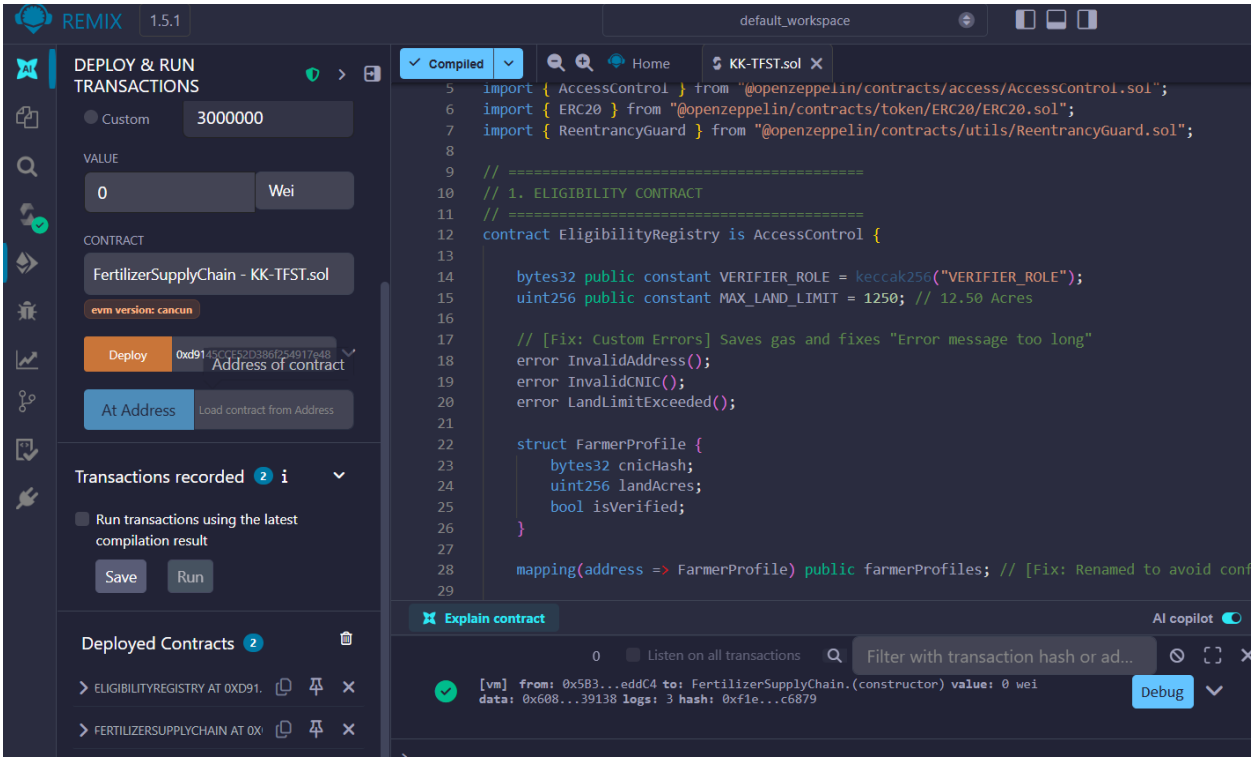
1. **Deploy EligibilityRegistry:**
    - Go to the **Deploy & Run Transactions** tab.
    - Ensure **Environment** is "Remix VM (Cancun)" (or similar).
    - In the **Contract** dropdown, select EligibilityRegistry.
    - Click **Deploy**.



- *Action:* In the **Deployed Contracts** section at the bottom, find the new contract and click the **Copy Address** icon (small squares) next to it.

0xd9145CCE52D386f254917e481eB44e9943F39138

2. **Deploy FertilizerSupplyChain:**
    - In the **Contract** dropdown, select FertilizerSupplyChain.
    - **Paste** the address you just copied into the _eligibilityContractAddress box next to the Deploy button.
    - Click **Deploy**.



Step 3: The Testing Workflow

To verify the PoC, we will simulate three actors using the accounts provided by Remix.

- **Account 1 (0x5B3...):** Government/Admin (The Deployer)
- **Account 2 (0xAb8...):** Farmer
- **Account 3 (0x4B2...):** Dealer

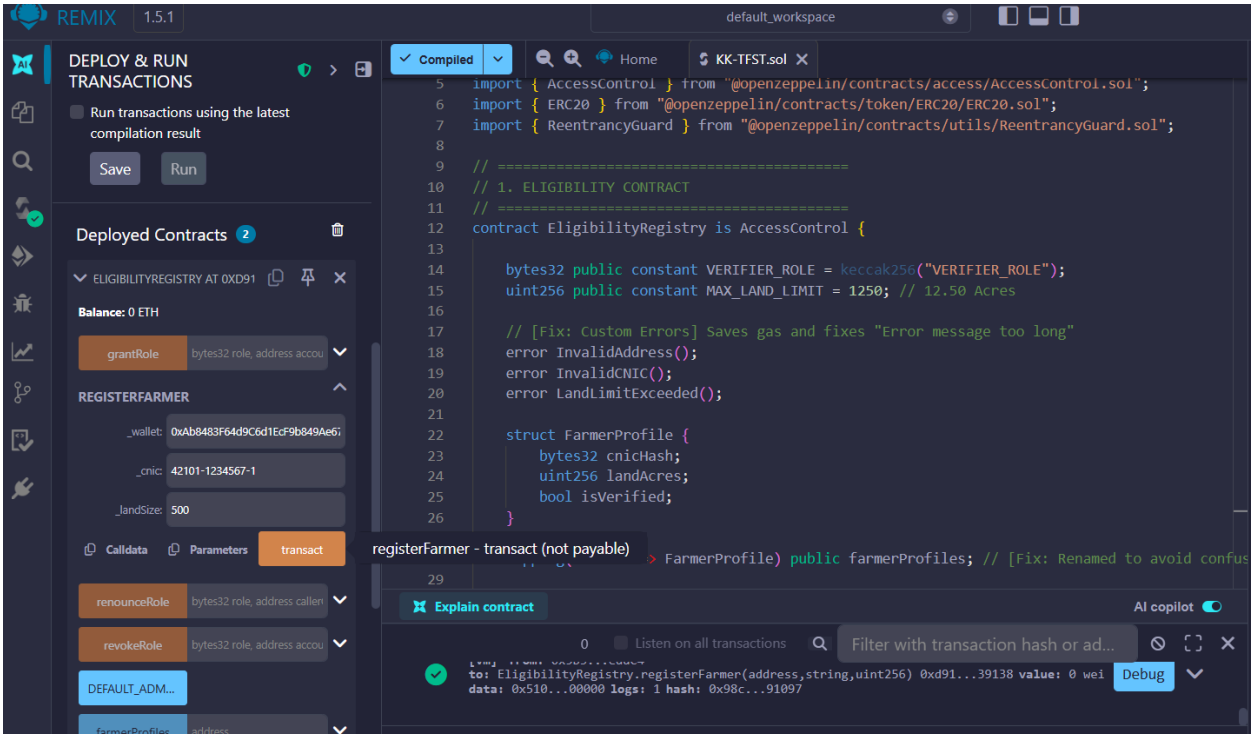
Step 1: Register the Farmer (Admin Action)

Before receiving a subsidy, the farmer must be in the registry.

1. Ensure **Account 1** is selected.
2. Expand the **ELIGIBILITYREGISTRY** contract.
3. Find registerFarmer.
4. **Input:**
    - _wallet: Copy/Paste **Account 2** address.
    - _cnic: "42101-1234567-1" (Any string).

- _landSize: 500 (Must be < 1250).

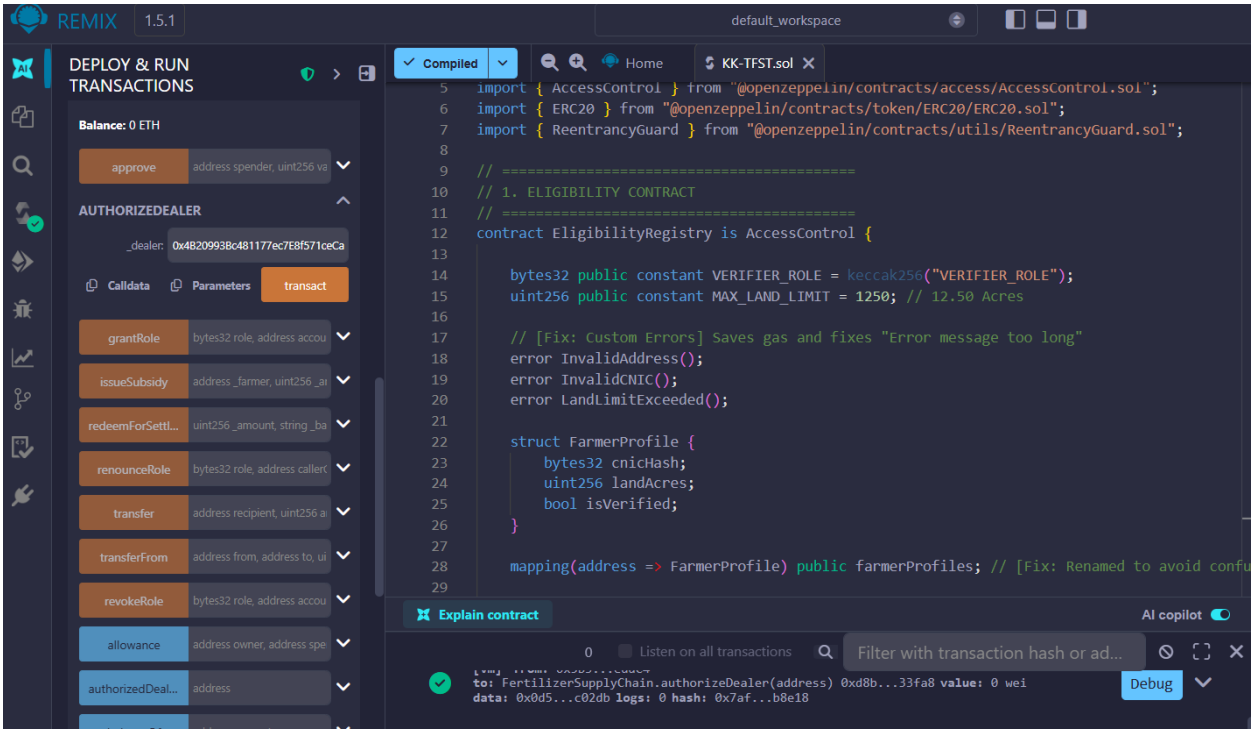
5. Click **Transact**.



Step 4: Authorize the Dealer (Admin Action)

The government authorizes a dealer to accept tokens.

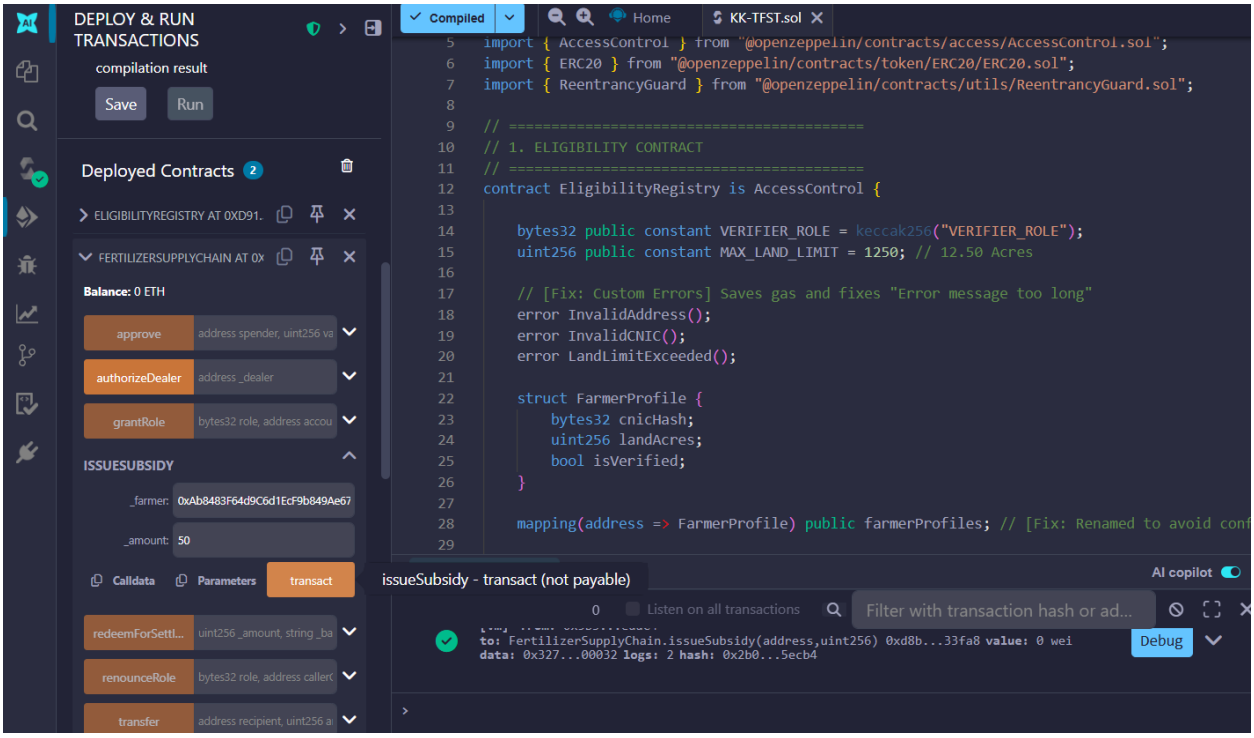
1. Expand the **FERTILIZERSUPPLYCHAIN** contract.
2. Find authorizeDealer.
3. **Input:**
    - _dealer: Copy/Paste **Account 3** address.
4. Click **Transact**.

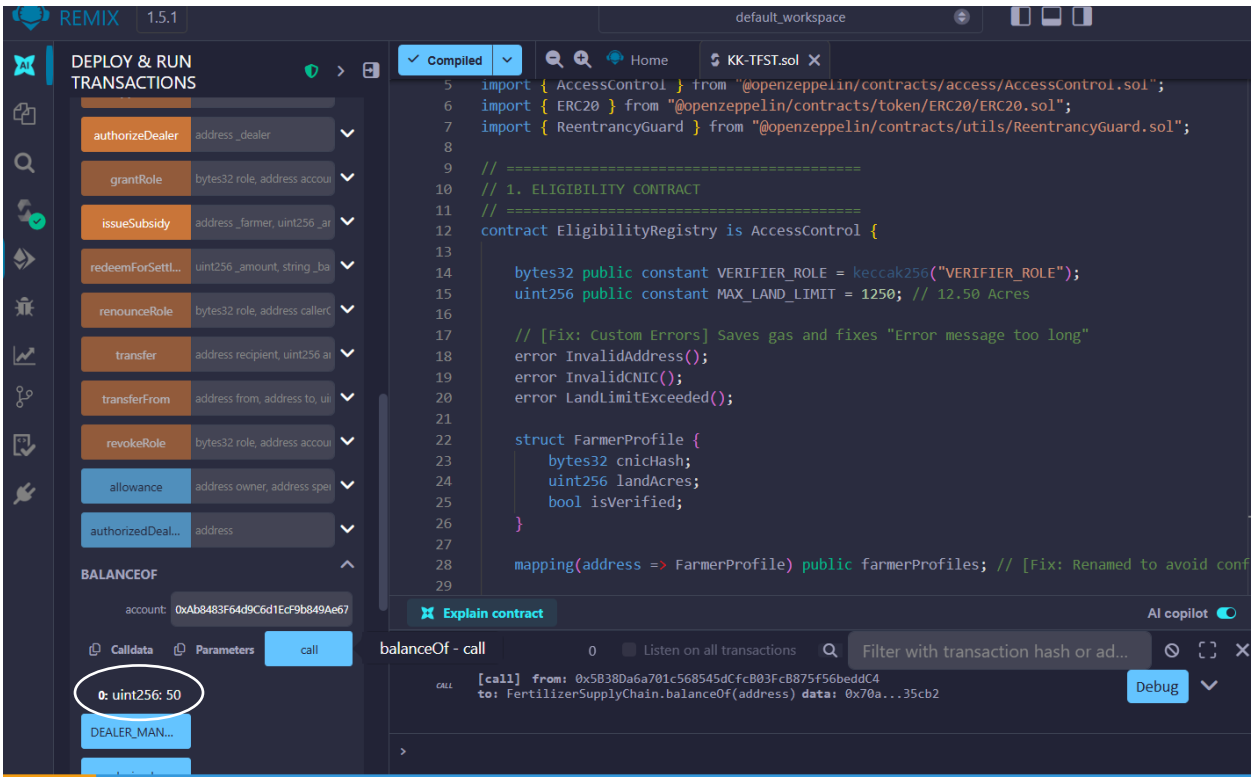


Step 5: Issue Subsidy (Admin Action)

The government mints tokens to the verified farmer.

1. Find issueSubsidy in the Supply Chain contract.
2. **Input:**
    - _farmer: Paste **Account 2** address.
    - _amount: 50 (Represents 50 tokens/bags).
3. Click **Transact**.
    - *Verification:* You can verify this by checking balanceOf for Account 2; it should now be 50.



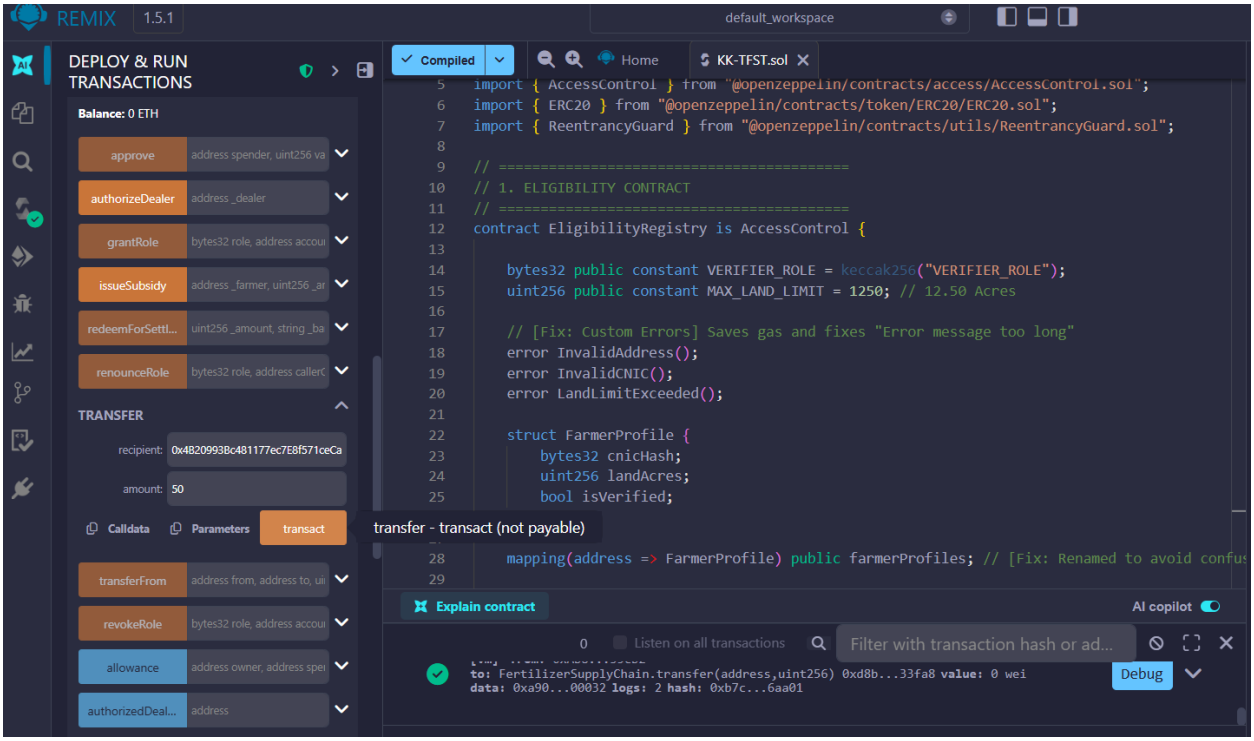


Step 6: Farmer Buys Fertilizer (Farmer Action)

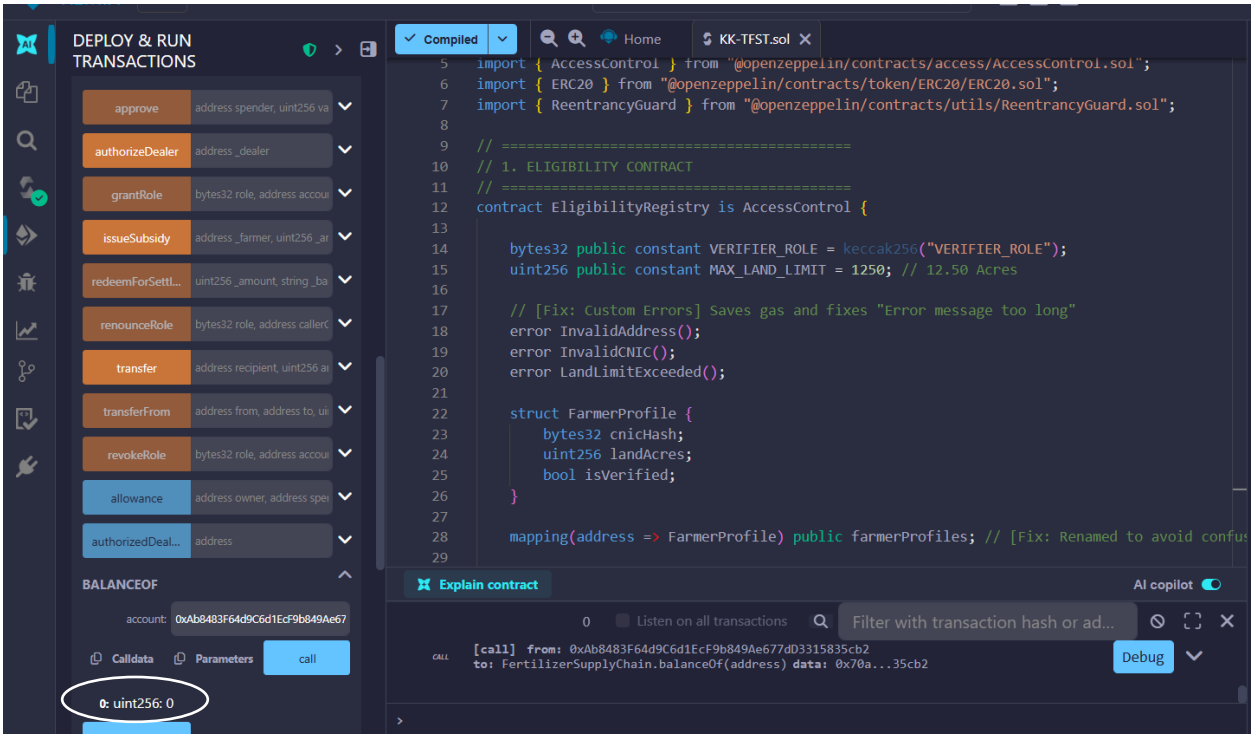
The farmer transfers tokens to the dealer in exchange for physical bags.

1. **Switch Account:** Change the top "Account" dropdown to **Account 2 (Farmer)**.

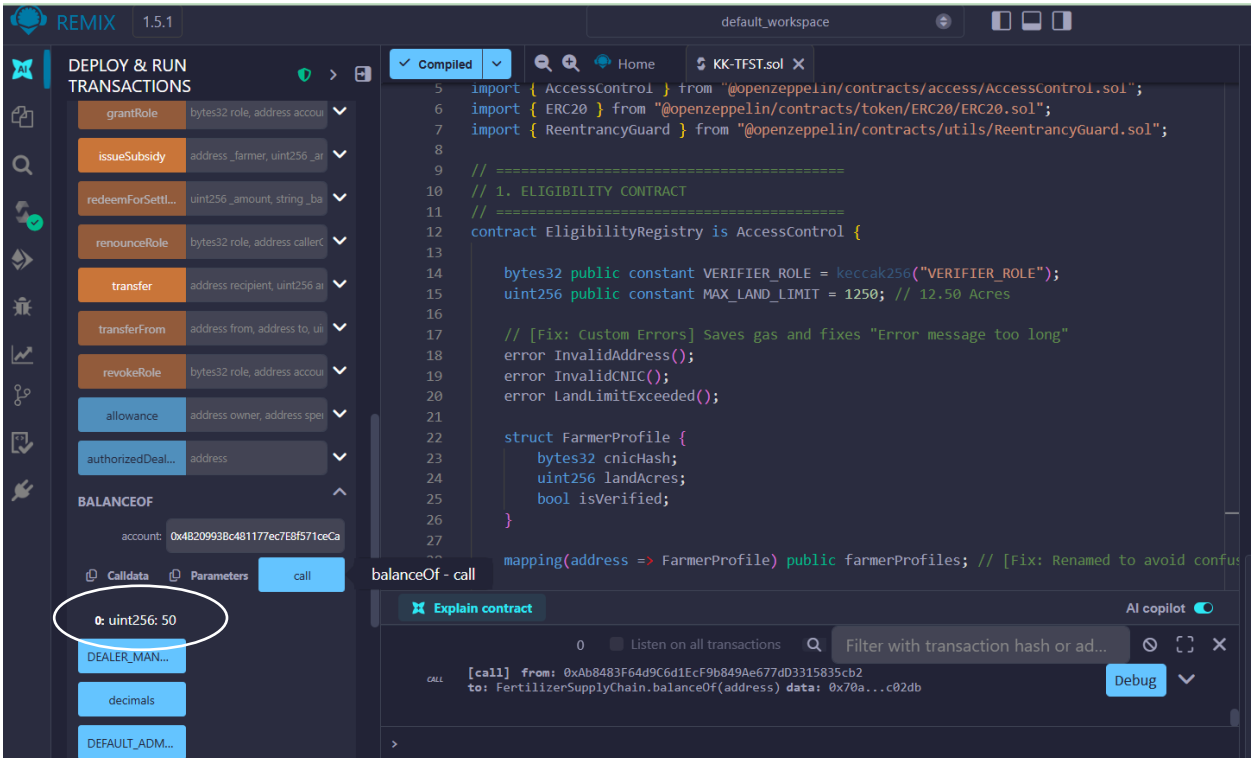
2. Find the transfer function (standard ERC20 function).
3. **Input:**
    - recipient: Paste **Account 3 (Dealer)** address.
    - amount: 50 (Buying 50 bags).
4. Click **Transact**.
    - *Note:* This transaction will fail if the recipient is not an authorized dealer.



After the transaction, the balance in farmer's wallet is now 0, means that the farmer has redeemed his/her tokens with the authorized dealer



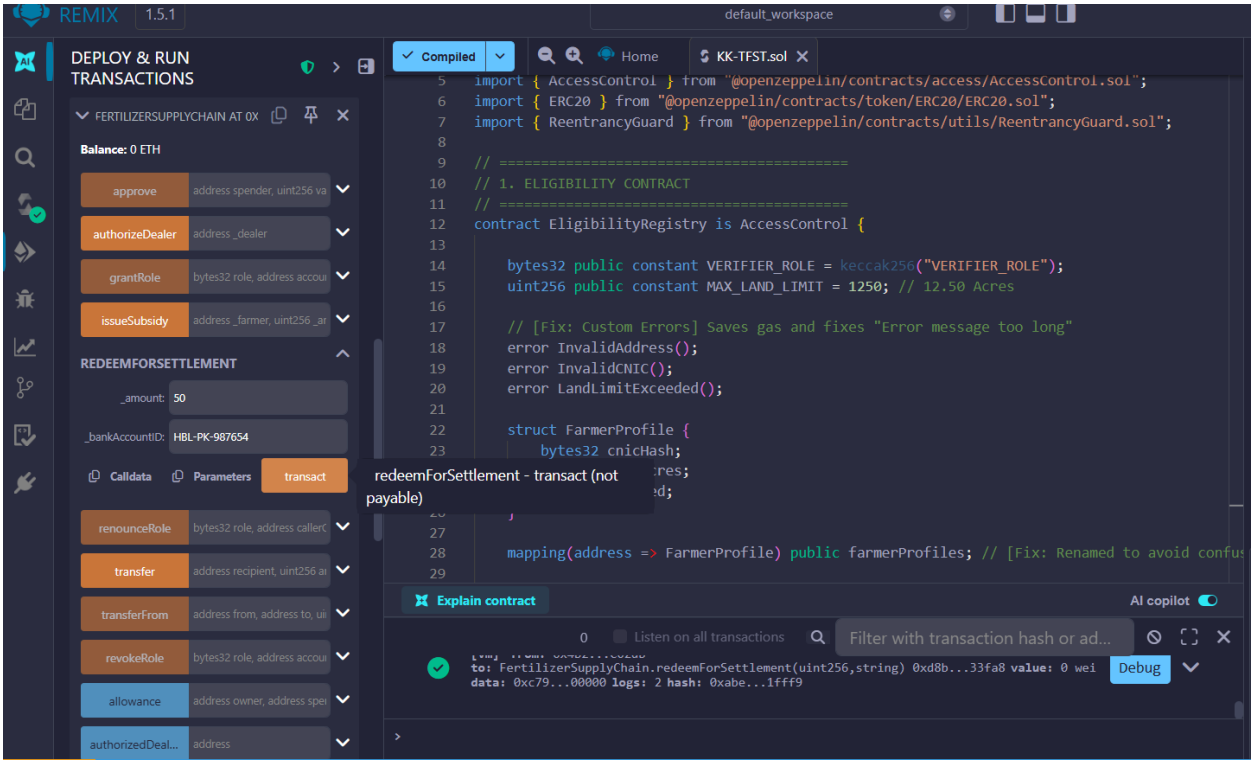
Conversely, the balance of the authorized dealer's account has increased to 50



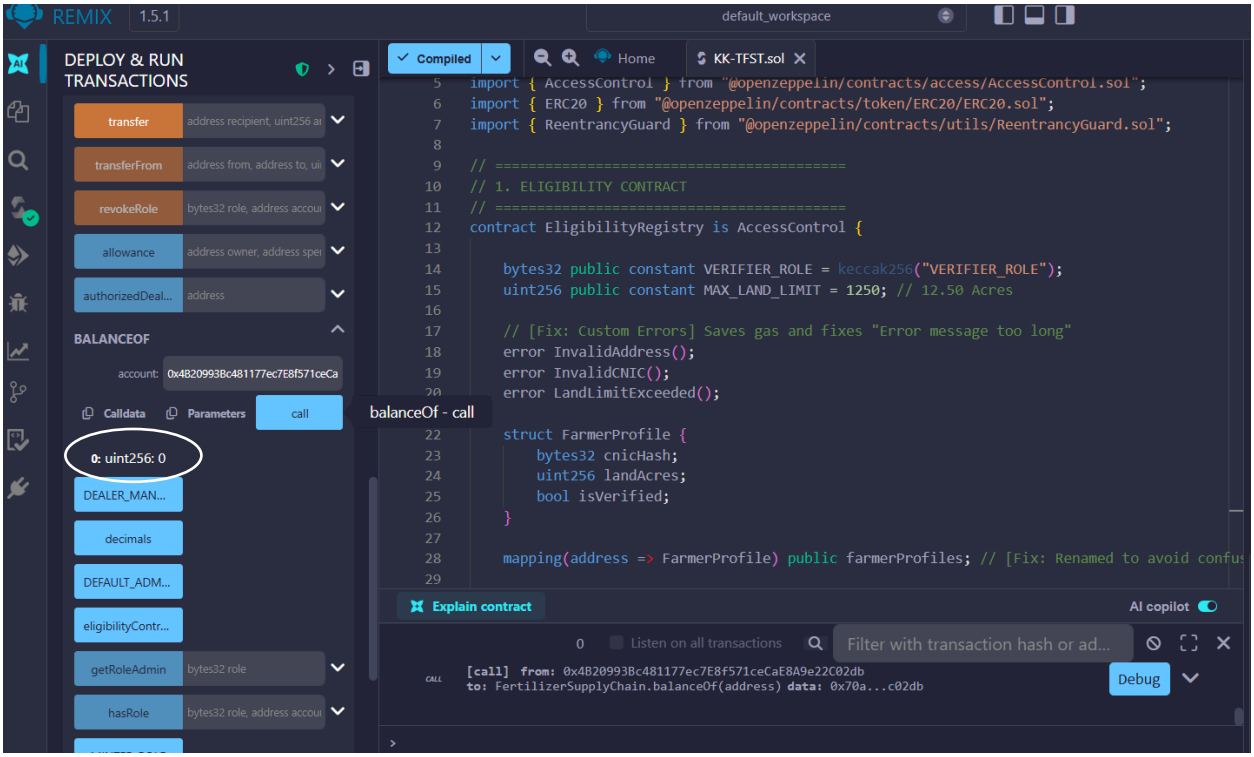
Step 7: Dealer Redemption (Dealer Action)

The dealer burns tokens to claim fiat money.

1. **Switch Account:** Change the top "Account" dropdown to **Account 3 (Dealer)**.
2. Find redeemForSettlement.
3. **Input:**
    - _amount: 50 (Redeeming the collected tokens).
    - _bankAccountID: "HBL-PK-987654".
4. Click **Transact**.



Result: The tokens are burned (removed from circulation), and the SettlementTriggered event is logged in the console with the bank details for the government to process the payment. Post burning, the balance of dealer's wallet has again fallen to zero, indicating successful burn.


REMIX 1.5.1
default_workspace
DEPLOY & RUN TRANSACTIONS
transfer
transferFrom
revokeRole
allowance
authorizedDeal...
BALANCEOF
account: 0x4B20993Bc481177ec7E8f571ceCa
Calldata
Parameters
call
balanceOf - call
0: uint256: 0
DEALER_MAN...
decimals
DEFAULT_ADM...
eligibilityContr...
getRoleAdmin
hasRole
Compiled
Home
KK-TFST.sol
import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
// ==
// 1. ELIGIBILITY CONTRACT
// ==
contract EligibilityRegistry is AccessControl {
bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
// [Fix: Custom Errors] Saves gas and fixes "Error message too long"
error InvalidAddress();
error InvalidCNIC();
error LandLimitExceeded();
struct FarmerProfile {
bytes32 cnicHash;
uint256 landAcres;
bool isVerified;
}
mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confu
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
[call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to: FertilizerSupplyChain.balanceOf(address) data: 0x70a...c02db
Debug

****

Annexure: Smart Contract Code for Remix

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

// [Fix: Named Imports] explicitly importing symbols reduces compiler ambiguity
import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

// ========================================
// 1. ELIGIBILITY CONTRACT
// ========================================
contract EligibilityRegistry is AccessControl {

    bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
    uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres

    // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
    error InvalidAddress();
    error InvalidCNIC();
    error LandLimitExceeded();

    struct FarmerProfile {
        bytes32 cnicHash;
        uint256 landAcres;
        bool isVerified;
```

```
}

mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confusion]

event FarmerVerified(address indexed farmer, uint256 landSize);

constructor() {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(VERIFIER_ROLE, msg.sender);
}

// [Fix: string calldata] Cheaper than 'memory' for external functions
function registerFarmer(address _wallet, string calldata _cnic, uint256 _landSize) external onlyRole(VERIFIER_ROLE) {
    if (_wallet == address(0)) revert InvalidAddress();
    if (bytes(_cnic).length == 0) revert InvalidCNIC();
    if (_landSize >= MAX_LAND_LIMIT) revert LandLimitExceeded();

    farmerProfiles[_wallet] = FarmerProfile({
        cnicHash: keccak256(abi.encodePacked(_cnic)),
        landAcres: _landSize,
        isVerified: true
    });

    emit FarmerVerified(_wallet, _landSize);
}
```

```
    // [Fix: Renamed argument] _farmerAddress distinguishes it from the mapping
    function isEligible(address _farmerAddress) external view returns (bool) {
        return farmerProfiles[_farmerAddress].isVerified;
    }
}

// ========================================
// 2. SUPPLY CHAIN & SETTLEMENT CONTRACT
// ========================================
contract FertilizerSupplyChain is ERC20, AccessControl, ReentrancyGuard {

    EligibilityRegistry public eligibilityContract;

    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant DEALER_MANAGER_ROLE =
keccak256("DEALER_MANAGER_ROLE");

    mapping(address => bool) public authorizedDealers;

    // [Fix: Custom Errors]
    error InvalidContractAddress();
    error InvalidDealerAddress();
    error InvalidFarmerAddress();
    error FarmerNotVerified();
    error UnauthorizedDealer();
```

```
    error InsufficientBalance();
    error InvalidBankDetails();
    error UnauthorizedCall();

    event SubsidyIssued(address indexed farmer, uint256 amount);
    event PhysicalBagHandover(address indexed farmer, address indexed dealer, uint256 bags);
    event SettlementTriggered(address indexed dealer, uint256 amountTokens, string bankDetails);

    constructor(address _eligibilityContractAddress) ERC20("Govt Fertilizer Subsidy", "GFS") {
        if (_eligibilityContractAddress == address(0)) revert InvalidContractAddress();

        eligibilityContract = EligibilityRegistry(_eligibilityContractAddress);

        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
        _grantRole(DEALER_MANAGER_ROLE, msg.sender);
    }

    // --- GOVT FUNCTIONS ---

    function authorizeDealer(address _dealer) external onlyRole(DEALER_MANAGER_ROLE) {
        if (_dealer == address(0)) revert InvalidDealerAddress();
        authorizedDealers[_dealer] = true;
```

```
    }

    function issueSubsidy(address _farmer, uint256 _amount) external
onlyRole(MINTER_ROLE) {
        if (_farmer == address(0)) revert InvalidFarmerAddress();

        // [Fix: Checks-Effects-Interaction Pattern]
        // 1. Interaction (Read external data)
        bool isVerified = eligibilityContract.isEligible(_farmer);

        // 2. Check (Validate data)
        if (!isVerified) revert FarmerNotVerified();

        // 3. Effect (Update state / Mint)
        _mint(_farmer, _amount);
        emit SubsidyIssued(_farmer, _amount);
    }

    // --- FARMER FUNCTIONS ---

    function transfer(address recipient, uint256 amount) public override nonReentrant
returns (bool) {
        if (!authorizedDealers[recipient]) revert UnauthorizedDealer();

        bool success = super.transfer(recipient, amount);

        if(success) {
```

```
        emit PhysicalBagHandover(msg.sender, recipient, amount);
    }
    return success;
  }

  // --- DEALER FUNCTIONS ---

  function redeemForSettlement(uint256 _amount, string calldata _bankAccountID)
external nonReentrant {
    if (!authorizedDealers[msg.sender]) revert UnauthorizedCall();
    if (balanceOf(msg.sender) < _amount) revert InsufficientBalance();
    if (bytes(_bankAccountID).length == 0) revert InvalidBankDetails();

    _burn(msg.sender, _amount);
    emit SettlementTriggered(msg.sender, _amount, _bankAccountID);
  }
}
```