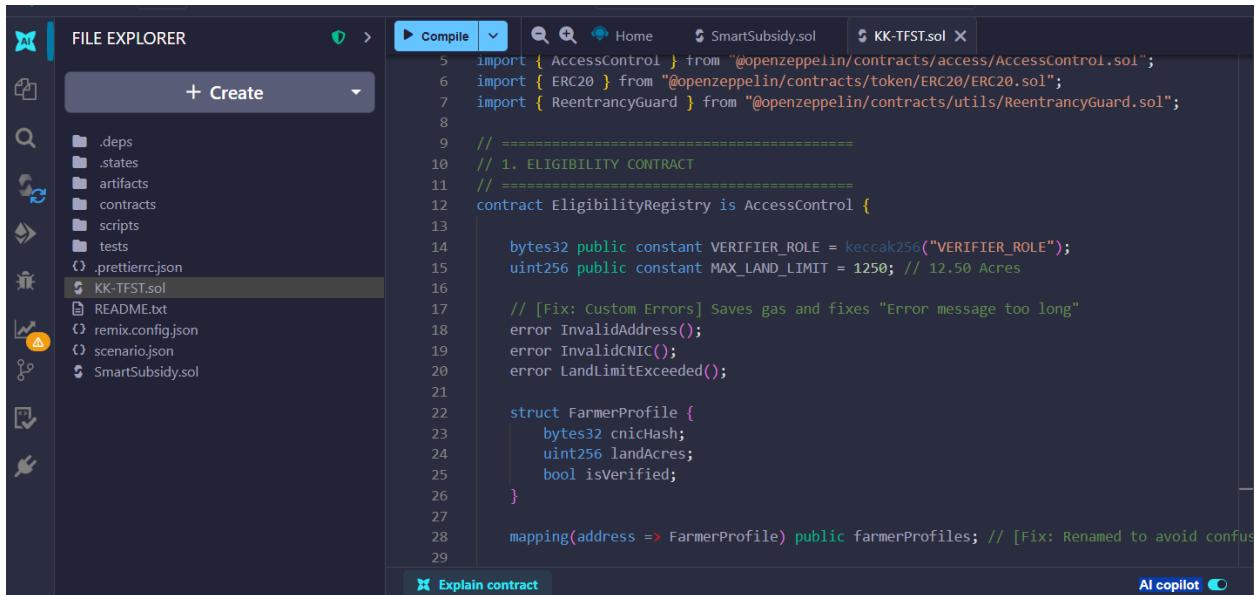


Supplement: Reproducing the TT-TFST Smart Subsidy Contract

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.



```
FILE EXPLORER          Compile ▾ Home SmartSubsidy.sol KK-TFST.sol
+ Create
.deps
.states
.artifacts
.contracts
.scripts
.tests
.prettierc.json
KK-TFST.sol
README.txt
remix.config.json
scenario.json
SmartSubsidy.sol

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confusion]
29 }
```

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.

```

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandlimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confusion]
29

```

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**.

```

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 conflict]
}

// Explain contract

```

Transactions recorded: Run transactions using the latest compilation result

Deployed Contracts: 0xd9145CCE52D386f254917e481eB44e9943F39138

- 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**.

```

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 conflict]
}

[vm] from: 0x583...eddC4 to: FertilizerSupplyChain.(constructor) value: 0 wei
data: 0x608...39138 logs: 3 hash: 0xf1e...c6879

```

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 (0xA8b...):** 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:**
 - o `_wallet`: Copy/Paste **Account 2** address.
 - o `_cnic`: "42101-1234567-1" (Any string).

- o `_landSize: 500` (Must be < 1250).

5. Click Transact.

```

REMIX 1.5.1 | default_workspace | KK-TFST.sol
DEPLOY & RUN TRANSACTIONS
Run transactions using the latest compilation result
Save Run
Deployed Contracts 2
ELIGIBILITYREGISTRY AT 0xD91...
Balance: 0 ETH
grantRole bytes32 role, address account
REGISTERFARMER
_wallet: 0xAb8483F64d9C6d1Ec9b849Ae6
_cnic: 42101-1234567-1
_landSize: 500
Calldata Parameters
registerFarmer - transact (not payable) FarmerProfile public farmerProfiles; // [Fix: Renamed to avoid confusion]
renounceRole bytes32 role, address caller
revokeRole bytes32 role, address account
DEFAULT_ADMIN...
farmerProfiles address

```

`contract EligibilityRegistry is AccessControl {`

- 5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
- 6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
- 7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
- 8
- 9 // =====
- 10 // 1. ELIGIBILITY CONTRACT
- 11 // =====
- 12 contract EligibilityRegistry is AccessControl {
- 13
- 14 bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
- 15 uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
- 16
- 17 // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
- 18 error InvalidAddress();
- 19 error InvalidCNIC();
- 20 error LandLimitExceeded();
- 21
- 22 struct FarmerProfile {
- 23 bytes32 cnicHash;
- 24 uint256 landAcres;
- 25 bool isVerified;
- 26 }
- 27
- 28 function registerFarmer(FarmerProfile memory farmerProfile) external {
- 29 require(msg.sender == VERIFIER_ROLE, "Only verifier can register farmer");
- 30 require(farmerProfile.landAcres <= MAX_LAND_LIMIT, "Land limit exceeded");
- 31 farmerProfiles.push(farmerProfile);
- 32 }
- 33 }

Explain contract

0 Listen on all transactions Filter with transaction hash or address

to: EligibilityRegistry.registerFarmer(address, string, uint256) 0xd91...39138 value: 0 wei Debug

data: 0x510...0000 logs: 1 hash: 0x98c...91097

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:**
 - o `_dealer: Copy/Paste Account 3 address.`
4. Click **Transact**.

The screenshot shows the REMIX IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists various functions with their parameters. On the right, the main area displays the `KK-TFST.sol` source code, which includes imports for AccessControl, ERC20, and ReentrancyGuard, and defines an `EligibilityRegistry` contract with a FarmerProfile struct and a mapping of addresses to FarmerProfiles.

```

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid conflict]
29

```

Step 5: Issue Subsidy (Admin Action)

The government mints tokens to the verified farmer.

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

The screenshot shows the Truffle UI interface. On the left, the 'Deploy & Run TRANSACTIONS' section lists several functions: approve, authorizeDealer, grantRole, issueSubsidy, redeemForSettle, renounceRole, transfer, and authorizedDealer. The 'issueSubsidy' function is selected. On the right, the 'Deployed Contracts' section shows the 'FERTILIZERSUPPLYCHAIN AT 0xd91...' contract. It has a balance of 0 ETH. The 'ISSUESUBSIDY' section shows a transaction history with one entry: 'issueSubsidy - transact (not payable)' to 'FertilizerSupplyChain.issueSubsidy(address,uint256) 0xd8b...33fa8 value: 0 wei'. The transaction details show '_farmer: 0xAb8483F64d9C6d1EcF9b849Ae67' and '_amount: 50'.

```

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid conflict]
29

```

The screenshot shows the REMIX IDE interface. On the left, the 'Deploy & Run TRANSACTIONS' section lists several functions: authorizeDealer, grantRole, issueSubsidy, redeemForSettle, renounceRole, transfer, transferFrom, revokeRole, allowance, and authorizedDealer. The 'issueSubsidy' function is selected. On the right, the 'Explain contract' section shows a transaction history with one entry: 'balanceOf - call' to 'FertilizerSupplyChain.balanceOf(address)' with a value of 0. The transaction details show 'from: 0x5838Da6a701c568545dCfcB03FcB875f56beddC4' and 'data: 0x70a...35cb2'.

```

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid conflict]
29

```

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:

- o recipient: Paste **Account 3 (Dealer)** address.
- o amount: 50 (Buying 50 bags).

4. Click **Transact**.

- o Note: This transaction will fail if the recipient is not an authorized dealer.

The screenshot shows the REMIX IDE interface with the following details:

- Contract:** KK-TFST.sol
- Code View:** Compiled view of the Solidity code, showing imports for AccessControl, ERC20, and ReentrancyGuard, and the implementation of an Eligibility Registry contract with a FarmerProfile struct and a mapping of FarmerProfiles.
- Deploy & Run Transactions:**
 - DEPLOY & RUN TRANSACTIONS:** Shows a balance of 0 ETH.
 - TRANSFER:** Set recipient to 0x4B20993Bc4B1177ec7E8f571ceCa and amount to 50.
 - Calldata Parameters:** Set to "transfer - transact (not payable)"
 - Call Data:** Set to "transferFrom", "revokeRole", "allowance", and "authorizedDeal...".
- Logs:** A log entry is shown for the transfer function call:
 - to: FertilizerSupplyChain.transfer(address,uint256) 0xd8b...33fa8
 - value: 0 wei
 - data: 0xa90...00032
 - logs: 2
 - hash: 0xb7c...6aa01

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

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with various icons. The main area has a title "DEPLOY & RUN TRANSACTIONS". Below it is a list of transactions with dropdown menus. One transaction, "balanceOf", is highlighted. The "call" button is pressed, and the result is displayed as "0: uint256: 0". On the right, the Solidity code for "KK-TFST.sol" is shown. The code defines a contract "EligibilityRegistry" that interacts with an "AccessControl" contract. It includes functions for approving, authorizing dealers, granting roles, issuing subsidies, redeeming for settlements, renouncing roles, transferring tokens, and allowing. It also defines a "FarmerProfile" struct and a mapping of addresses to profiles. The "balanceOf" function is part of the "farmerProfiles" mapping.

```

5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnicHash;
24         uint256 landAcres;
25         bool isVerified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confusion]
29

```

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

This screenshot shows the REMIX IDE interface. The left sidebar has icons for file operations, search, and deployment. The main area has a title "DEPLOY & RUN TRANSACTIONS". A list of transactions is shown, with "balanceOf" being the selected one. The "call" button is pressed, and the result is displayed as "0: uint256: 50". On the right, the Solidity code for "KK-TFST.sol" is displayed, which is identical to the one in the previous screenshot. It includes the "EligibilityRegistry" contract with its various functions and the "farmerProfiles" mapping.

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:**

- o `_amount: 50` (Redeeming the collected tokens).
- o `_bankAccountID: "HBL-PK-987654"`.

4. Click **Transact**.

The screenshot shows the REMIX IDE interface with the following details:

- Contract:** KK-TFST.sol
- Workspace:** default_workspace
- Deploy & Run Transactions:**
 - FERTILIZERSUPPLYCHAIN AT 0x**:
 - `approve`: address spender, uint256 value
 - `authorizeDealer`: address _dealer
 - `grantRole`: bytes32 role, address account
 - `issueSubsidy`: address _farmer, uint256 amount
 - REDEEMFORSETTLEMENT**:
 - `_amount: 50`
 - `_bankAccountID: HBL-PK-987654`
- Code View:** Shows the Solidity code for the `EligibilityRegistry` contract, including imports for AccessControl, ERC20, and ReentrancyGuard, and a mapping of FarmerProfile to address.
- Logs:** Shows a log entry for the `redeemForSettlement` function being called with `to: FertilizerSupplyChain.redeemForSettlement(uint256,string) 0xd8b...33fa8 value: 0 wei` and `data: 0xc79...00000 logs: 2 hash: 0xaebe...fffff`.

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.

The screenshot shows the REMIX IDE interface with the following details:

Left Panel (DEPLOY & RUN TRANSACTIONS):

- Shows several transaction buttons:
 - transfer: address recipient, uint256 amount
 - transferFrom: address from, address to, uint256 amount
 - revokeRole: bytes32 role, address account
 - allowance: address owner, address spender, uint256 amount
 - authorizedDeal...: address
- BALANCEOF section:
 - account: 0x4B20993Bc481177ec7E8f571ceCa
 - Calldata, Parameters, call buttons
 - Value input field: 0: uint256: 0 (circled)
 - Buttons: DEALER_MANAGER, decimals, DEFAULT_ADMIN, eligibilityContr..., getRoleAdmin, hasRole

Right Panel (Contract View):

- Compiled tab selected.
- Contract: KK-TFST.sol
- Contract code:

```
5 import { AccessControl } from "@openzeppelin/contracts/access/AccessControl.sol";
6 import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7 import { ReentrancyGuard } from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
8
9 // =====
10 // 1. ELIGIBILITY CONTRACT
11 // =====
12 contract EligibilityRegistry is AccessControl {
13
14     bytes32 public constant VERIFIER_ROLE = keccak256("VERIFIER_ROLE");
15     uint256 public constant MAX_LAND_LIMIT = 1250; // 12.50 Acres
16
17     // [Fix: Custom Errors] Saves gas and fixes "Error message too long"
18     error InvalidAddress();
19     error InvalidCNIC();
20     error LandLimitExceeded();
21
22     struct FarmerProfile {
23         bytes32 cnichash;
24         uint256 landAcres;
25         bool isverified;
26     }
27
28     mapping(address => FarmerProfile) public farmerProfiles; // [Fix: Renamed to avoid confusion]
29 }
```
- Explain contract button.
- Call details:
 - CALL [call] from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: FertilizerSupplyChain.balanceOf(address) data: 0x70a...c02db
- AI copilot button.

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);

}
```