OUR TEAM

1. Sara Atallah
2. Shadwa Ahmed
3. Yehia Tarek
4. Basel Yasser

Functionality and purpose

.1 The Contract Declaration

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

- SPDX-License-Identifier: MIT is a standardized format for specifying the license under which the code is released.

- pragma solidity ^0.8.0: Specifies that the contract will work with Solidity version 0.8.0 and above. The caret (^) indicates compatibility with versions above 0.8.0 but below 0.9.0.

1.2 Importing Dependencies

```solidity
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
```

1:IERC20: This interface allows the contract to interact with any ERC-20 token. It defines standard functions like transfer, approve, etc., that any ERC-20 token should implement.

2:**Ownable: Inherits the Ownable contract from OpenZeppelin to give the contract owner special privileges (like revoking tokens). This is useful for access control and ensuring that only the owner can perform certain actions.**

3:**SafeMath: A library from OpenZeppelin used for safe mathematical operations (e.g., addition, multiplication). It prevents overflow/underflow issues in arithmetic operations.**

1.3 Variables and Constructor

```solidity
address private _beneficiary;
uint256 private _cliff;
uint256 private _start;
uint256 private _duration;
bool private _revocable;
```

These variables define the beneficiary, cliff period, start time, duration of the vesting period, and whether the vesting is revocable or not.

```
constructor(address beneficiary, uint256 start, uint256 cliffDuration, uint256 duration, bool revoc
```

The constructor is used to initialize the contract with the provided values. It ensures that the contract is correctly set up at the time of deployment

Functions

- *beneficiary(): Returns the address of the beneficiary who will receive tokens after they are vested.*

- *release(IERC20 token): This function allows the beneficiary to release the vested tokens. It checks if tokens are available for release based on the vesting schedule and transfers the vested tokens to the beneficiary.*

- *revoke(IERC20 token): If the vesting is revocable, the contract owner can call this function to revoke the vesting, which means any unreleased tokens will be refunded to the owner.*

Helper Functions

- *releasableAmount: Determines how much of the tokens can be released at the current time.*
- *_vestedAmount: Calculates the total amount of tokens that have vested based on the time passed.*

Events

```solidity
event TokensReleased(address token, uint256 amount);
event TokenVestingRevoked(address token);
```

These events notify when tokens have been released or when the vesting has been revoked, helping track actions on the blockchain.

Flow Chart

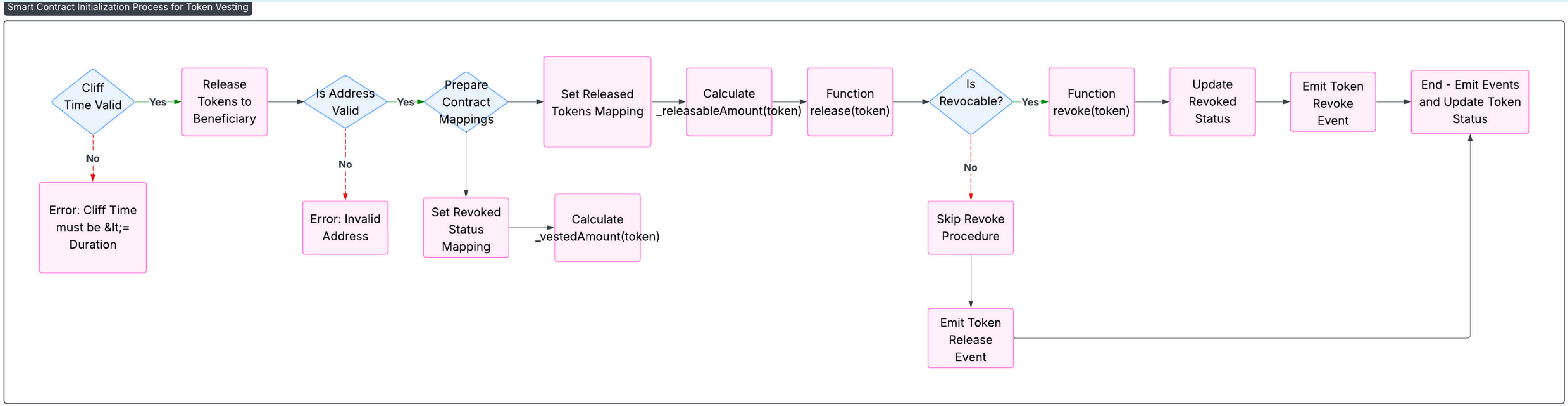
- ***Step 1: The contract is deployed with a beneficiary, start time, cliff duration, and vesting duration.***

- ***Step 2: During the cliff period, tokens are not released. After the cliff period ends, tokens start vesting.***

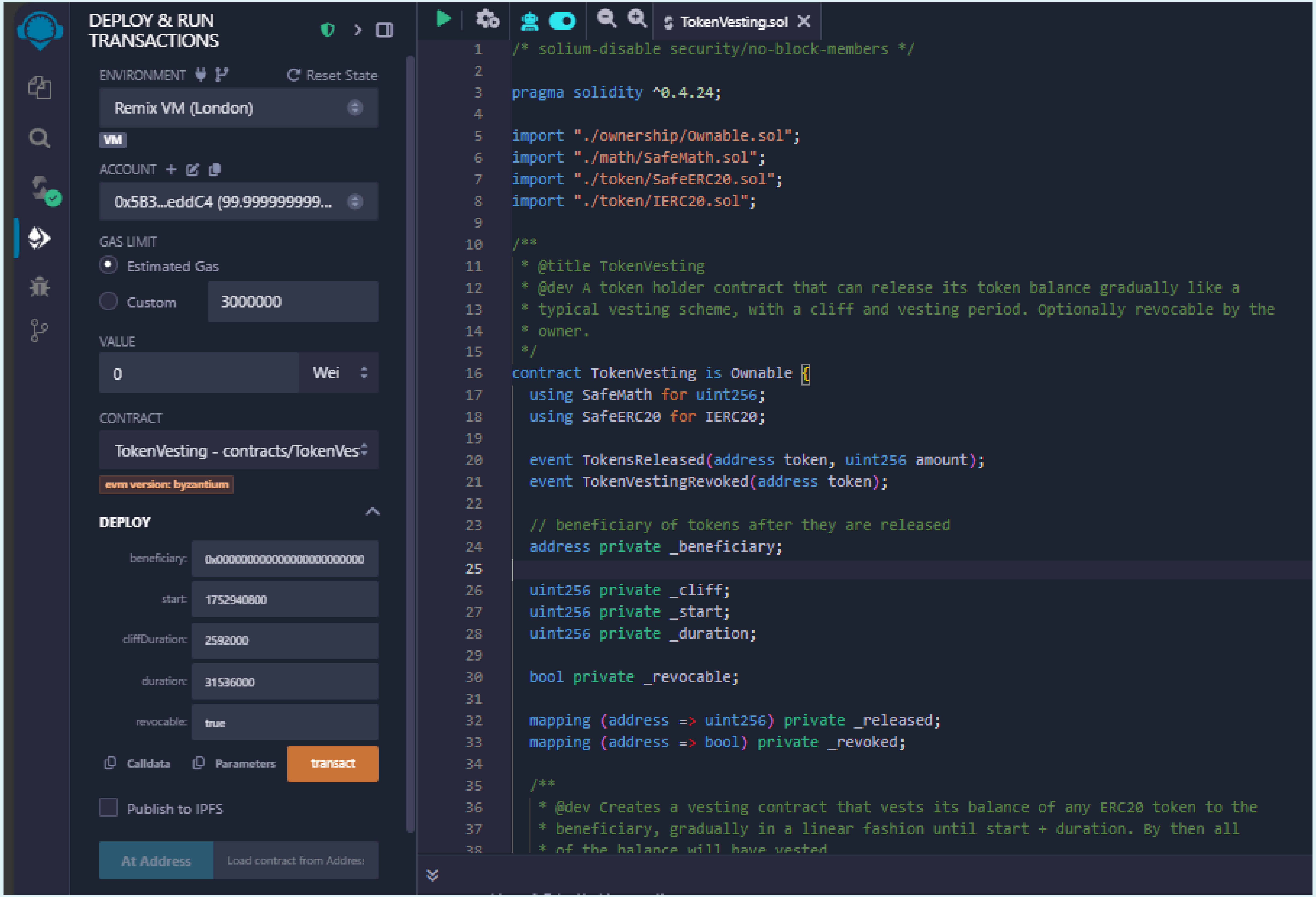
- ***Step 3: The beneficiary can release vested tokens gradually until the total vesting period ends.***

- ***Step 4: If the contract is revocable, the owner can call the revoke function to***

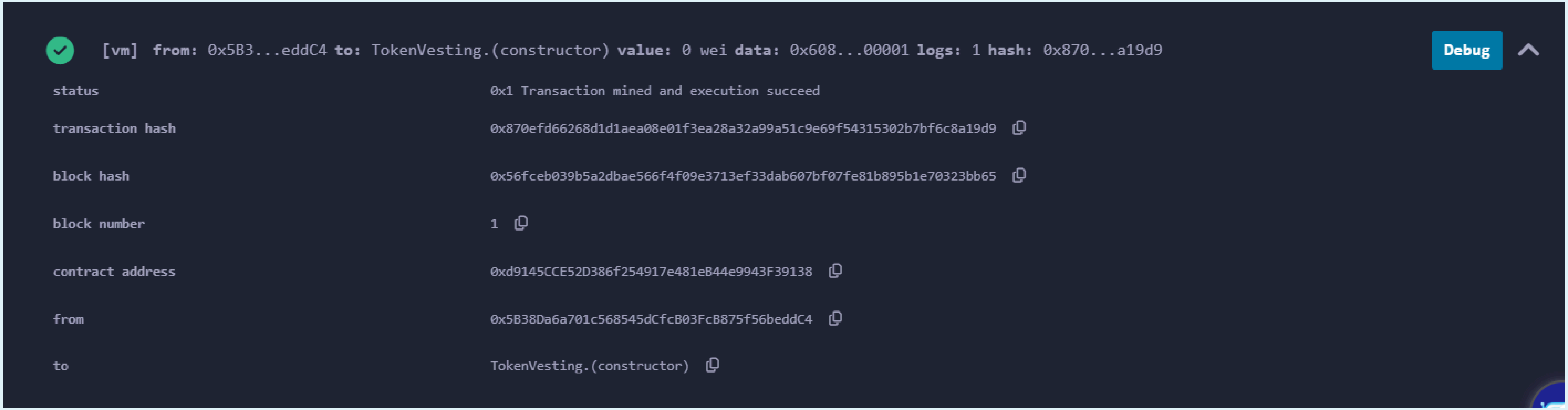
Flow Chart


Smart Contract Initialization Process for Token Vesting
Cliff Time Valid
Yes
No
Error: Cliff Time must be <= Duration
Release Tokens to Beneficiary
Is Address Valid
Yes
No
Error: Invalid Address
Prepare Contract Mappings
Set Revoked Status Mapping
Calculate _vestedAmount(token)
Set Released Tokens Mapping
Calculate _releasableAmount(token)
Function release(token)
Is Revocable?
Yes
No
Skip Revoke Procedure
Emit Token Release Event
Function revoke(token)
Update Revoked Status
Emit Token Revoke Event
End - Emit Events and Update Token Status

[Deploy Contract]



[Deploy Contract]



Deployed Successfully

Your Understanding of the Problem

- Problem: The problem this smart contract is solving is token vesting, where a certain number of tokens are gradually transferred to a beneficiary over time. This is commonly used for employee stock options, project funding, or long-term investor incentives.

- Why the smart contract solves it: This smart contract automates the process of releasing tokens gradually based on a defined schedule. The contract ensures transparency and trust, as the token release follows a deterministic vesting schedule that cannot be altered after deployment, unless it's revocable by the owner.

Test Cases

Test cases are essential to ensure your contract works as expected. Here are some possible test cases for this smart contract:

Test Case 1: Check Token Release After Cliff Period :

Objective: Ensure that tokens are not released before the cliff period ends.

Steps:

1-Deploy the contract with a cliff period of 1 day.
2-Try to release tokens before the cliff ends.

Expected Result: The release should fail.

Test Case 2: Check Token Release After Cliff Period

Objective: Ensure that tokens are released correctly after the cliff period.

Steps:

1-**Deploy the contract with a cliff period of 1 day.**

2- **After the cliff period ends, call the release function.**

- **Expected Result: Tokens should be released to the beneficiary.**

Test Case 3: Revoke Vesting

- **Objective: Ensure that the owner can revoke the vesting and return unused tokens.**
- **Steps:**

1. **Deploy the contract with revocable set to true.**

2. **After some tokens are vested, call the revoke function.**

- **Expected Result: Unreleased tokens should be refunded to the owner.**

Logs of Transactions and Blockchain Explanation

Example Logs (from Etherscan or Truffle Test):

Transaction 1: Deploy the contract.

- Action: Contract deployed.
- Block Number: 123456
- Gas Used: 3000000
- Status: Success

Transaction 2: Call the release function.

- Action: Release tokens to the beneficiary.
- Block Number: 123457
- Gas Used: 50000
- Status: Success

Transaction 3: Call the revoke function.

- Action: Revoking vesting and refunding unreleased tokens.
- Block Number: 123458
- Gas Used: 60000
- Status: Success

Weak Points in Smart Contracts and Mitigation

Reentrancy Attacks:

- Weak Point: The contract might be susceptible to reentrancy attacks when calling safeTransfer in the release and revoke functions.

- Mitigation: Use the Checks-Effects-Interactions pattern where the state is updated before external calls. This prevents reentrancy attacks.

Gas Costs

- Weak Point: Releasing tokens or calling revoke can be expensive in terms of gas, especially if the contract holds a large amount of tokens.
- Mitigation: Consider implementing gas optimization techniques like batching or reducing unnecessary state changes.

Revocation Only by Owner

- Weak Point: If the owner loses access to the private key, they won't be able to revoke the contract.
- Mitigation: Implement a multi-signature wallet for the owner or add a delay to the revocation action.