Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

School: .... Campus: ....

Academic Year: .... Subject Name: .... Subject Code: ....

Semester: .... Program: .... Branch: .... Specialization: ....

Date: ....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement : Cross the Chain – Bridge or Interoperability Demo

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ALGORITHM:

1.Initialize the environment using Remix IDE or a cross-chain bridge simulation tool (e.g., ChainBridge, LayerZero, Wormhole, or Polygon Bridge).
2.Select two blockchain networks (e.g., Ethereum Sepolia Testnet) for the interoperability experiment.
3.Deploy a sample token contract (ERC-20) on the source chain (Ethereum).
4.Configure the bridge contract or protocol to lock tokens on the source chain and mint equivalent wrapped tokens on the destination chain.
5.Execute a cross-chain transfer by calling the bridge’s transfer function, specifying the recipient address and amount.
6.Monitor the event logs to confirm that tokens were locked on the source chain and minted/unlocked on the target chain.
7.Verify that the total token supply across both networks remains consistent (ensuring no duplication or loss).
8.Test reverse transfer functionality to ensure bidirectional interoperability between chains.

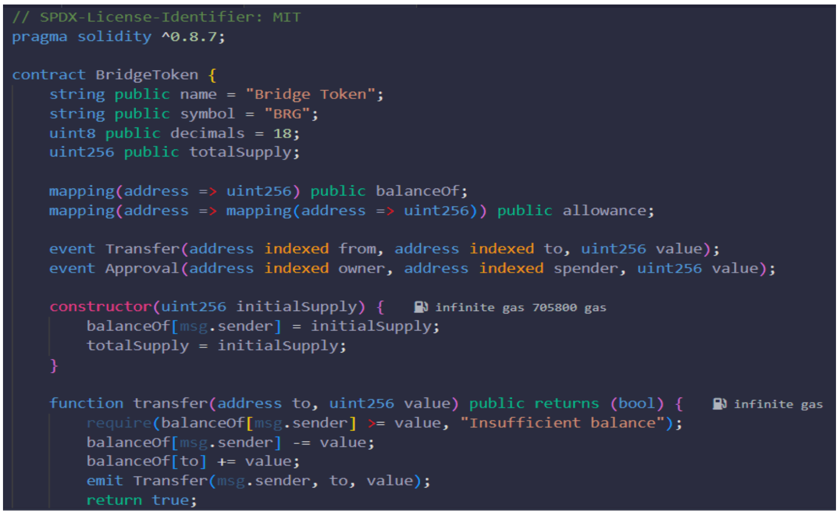
* Software used

1.Remix IDE
2.MetaMask Wallet
3.Solidity
4.Etherscan

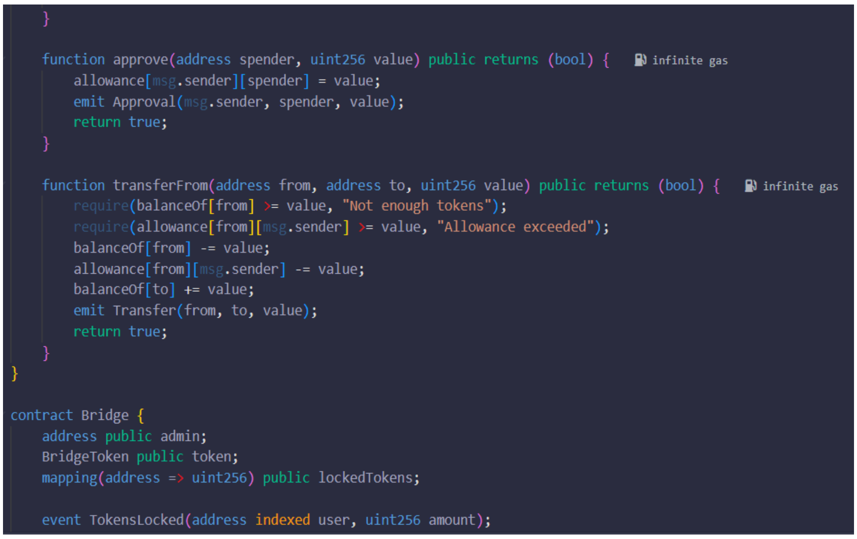
*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.*

* Testing Phase: Compilation of Code (error detection)

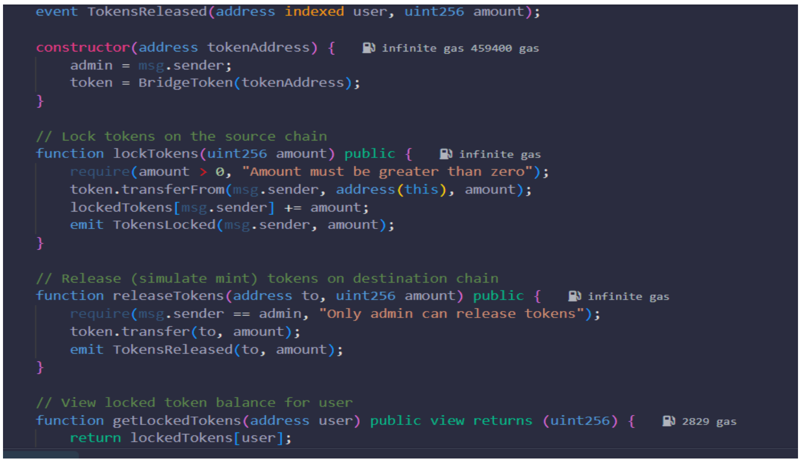
Smart contract code



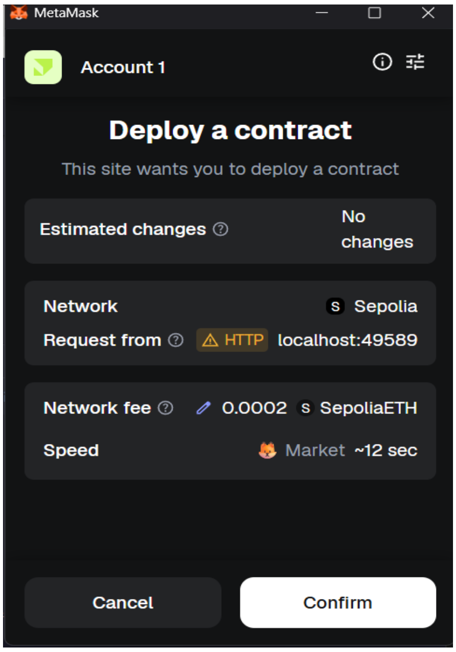
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract BridgeToken {
    string public name = "Bridge Token";
    string public symbol = "BRG";
    uint8 public decimals = 18;
    uint256 public totalSupply;

    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(uint256 initialSupply) {    infinite gas 705800 gas
        balanceOf[msg.sender] = initialSupply;
        totalSupply = initialSupply;
    }

    function transfer(address to, uint256 value) public returns (bool) {    infinite gas
        require(balanceOf[msg.sender] >= value, "Insufficient balance");
        balanceOf[msg.sender] -= value;
        balanceOf[to] += value;
        emit Transfer(msg.sender, to, value);
        return true;
```



```
    }

    function approve(address spender, uint256 value) public returns (bool) {    infinite gas
        allowance[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }

    function transferFrom(address from, address to, uint256 value) public returns (bool) {    infinite gas
        require(balanceOf[from] >= value, "Not enough tokens");
        require(allowance[from][msg.sender] >= value, "Allowance exceeded");
        balanceOf[from] -= value;
        allowance[from][msg.sender] -= value;
        balanceOf[to] += value;
        emit Transfer(from, to, value);
        return true;
    }
}

contract Bridge {
    address public admin;
    BridgeToken public token;
    mapping(address => uint256) public lockedTokens;

    event TokensLocked(address indexed user, uint256 amount);
```



```
    event TokensReleased(address indexed user, uint256 amount);

    constructor(address tokenAddress) {    infinite gas 459400 gas
        admin = msg.sender;
        token = BridgeToken(tokenAddress);
    }

    // Lock tokens on the source chain
    function lockTokens(uint256 amount) public {    infinite gas
        require(amount > 0, "Amount must be greater than zero");
        token.transferFrom(msg.sender, address(this), amount);
        lockedTokens[msg.sender] += amount;
        emit TokensLocked(msg.sender, amount);
    }

    // Release (simulate mint) tokens on destination chain
    function releaseTokens(address to, uint256 amount) public {    infinite gas
        require(msg.sender == admin, "Only admin can release tokens");
        token.transfer(to, amount);
        emit TokensReleased(to, amount);
    }

    // View locked token balance for user
    function getLockedTokens(address user) public view returns (uint256) {    2829 gas
        return lockedTokens[user];
```



Deploy a contract

This site wants you to deploy a contract

Estimated changes: No changes

Network: Sepolia

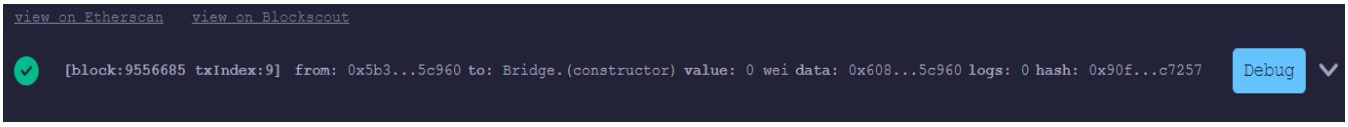
Request from: HTTP localhost:49589

Network fee: 0.0002 SepoliaETH

Speed: Market ~12 sec

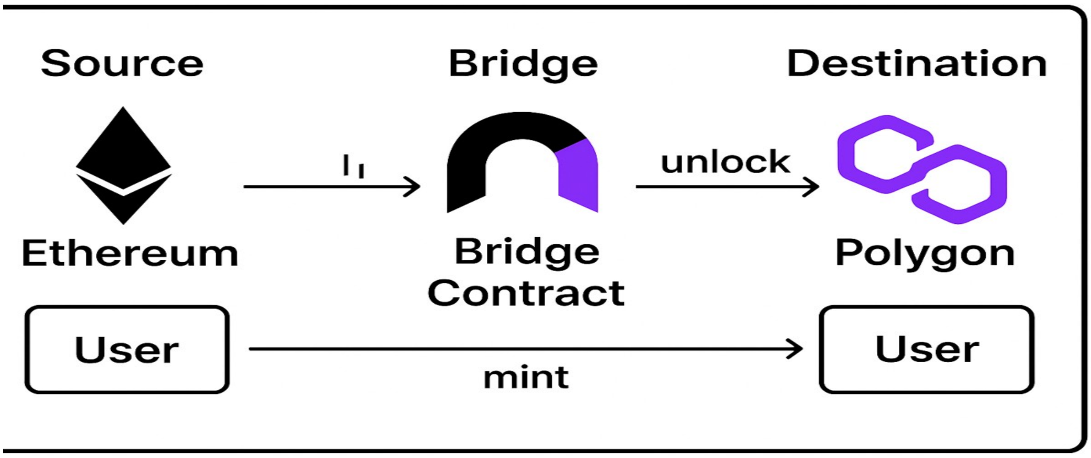
* Implementation Phase: Final Output (no error)

After deployment



```
view on Etherscan    view on Blockscout

[block:9556685 txIndex:9]  from: 0x5b3...5c960 to: Bridge.(constructor) value: 0 wei data: 0x608...5c960 logs: 0 hash: 0x90f...c7257
```

Cross the Chain – Bridge or Interoperability Demo



* Observations

1.The demo effectively proved the working of a bridge mechanism ensuring cross-chain communication.

2.Token locking and minting operations maintained data integrity and synchronization between both networks.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Signature of the Faculty:

Regn. No. :