EXPERIMENT 1

Aim: Study of simulation of blockchain

Theory:

SHA-256 (Secure Hash Algorithm 256-bit)

SHA-256 is part of the SHA-2 family of cryptographic hash functions. It takes an input (message or data) of any length and produces a **fixed 256-bit (32-byte) hash value**. **Deterministic**, meaning the same input always produces the same hash. Widely used in **blockchain, digital signatures, password hashing, and data integrity checks**. Designed to be **one-way** (cannot reverse the hash to get the original input) and collision-resistant (hard to find two inputs with the same hash).

Advantages of SHA-256

1. **Security:** Resistant to pre-image, second pre-image, and collision attacks.
2. **Fixed Output:** Always produces a 256-bit hash regardless of input size.
3. **Deterministic:** Same input always generates the same hash, ensuring consistency.
4. **Widely Adopted:** Supported in most cryptographic libraries and used in blockchain systems like Bitcoin.
5. **Integrity Verification:** Ensures data has not been altered.

Disadvantages of SHA-256

1. **Computationally Intensive:** Slower than simpler hashes like MD5 or SHA-1.
2. **Not Quantum-Proof:** Could be vulnerable to future quantum computing attacks.
3. **Irreversible:** Cannot retrieve the original input (though this is by design, it may be a limitation for some use cases like password recovery).
4. **Fixed Output Length:** Cannot produce variable-length hashes (use SHA-512 or SHA-3 for different sizes if needed).

Block

A block in a blockchain is a data structure that stores a collection of information, typically including transaction data, a timestamp, and a reference to the previous block via its hash. Each block contains:

1. Index/Height: Position of the block in the chain.
2. Timestamp: When the block was created.
3. Data/Transactions: The actual information being recorded.
4. Previous Hash: A hash of the previous block, linking blocks together.
5. Nonce (optional): Used in proof-of-work to satisfy mining difficulty.
6. Hash: The cryptographic hash of the current block's contents.

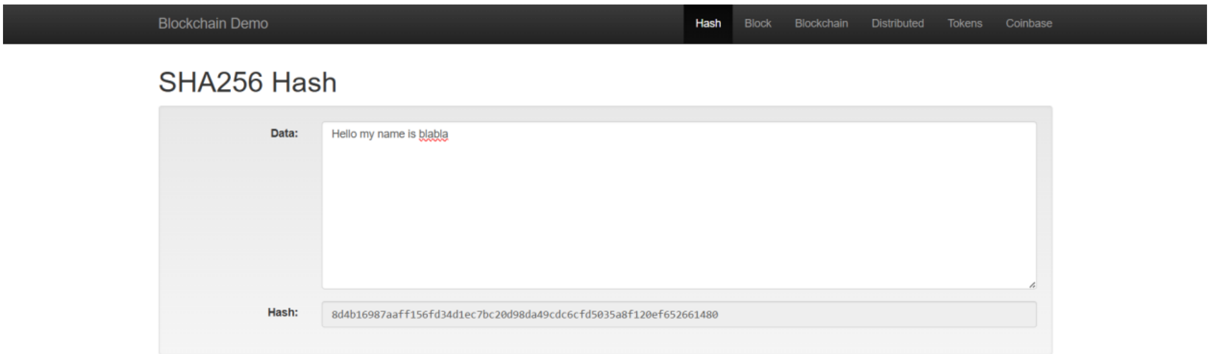
By linking blocks through hashes, each block ensures the integrity and immutability of the blockchain, making it resistant to tampering or fraud.

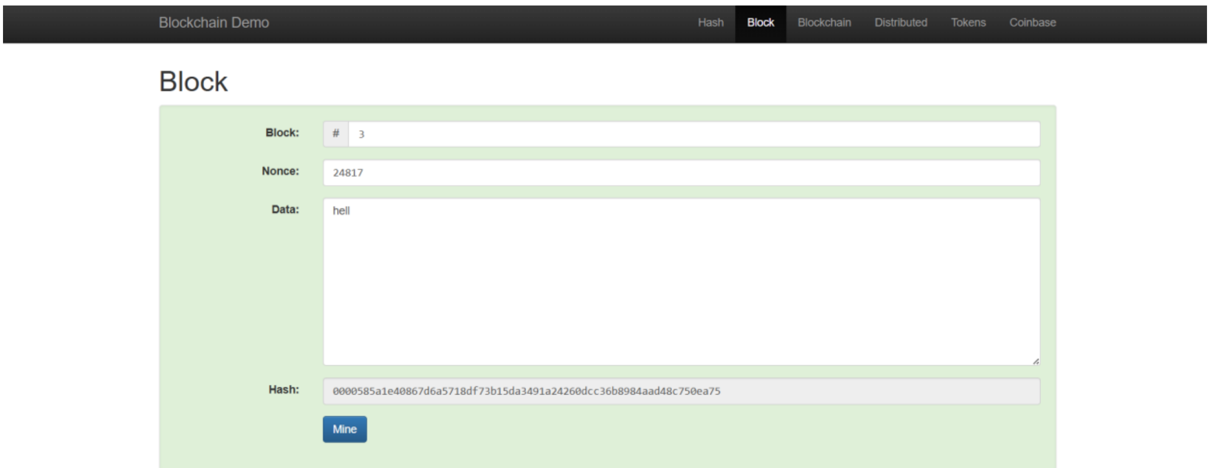
Blockchain

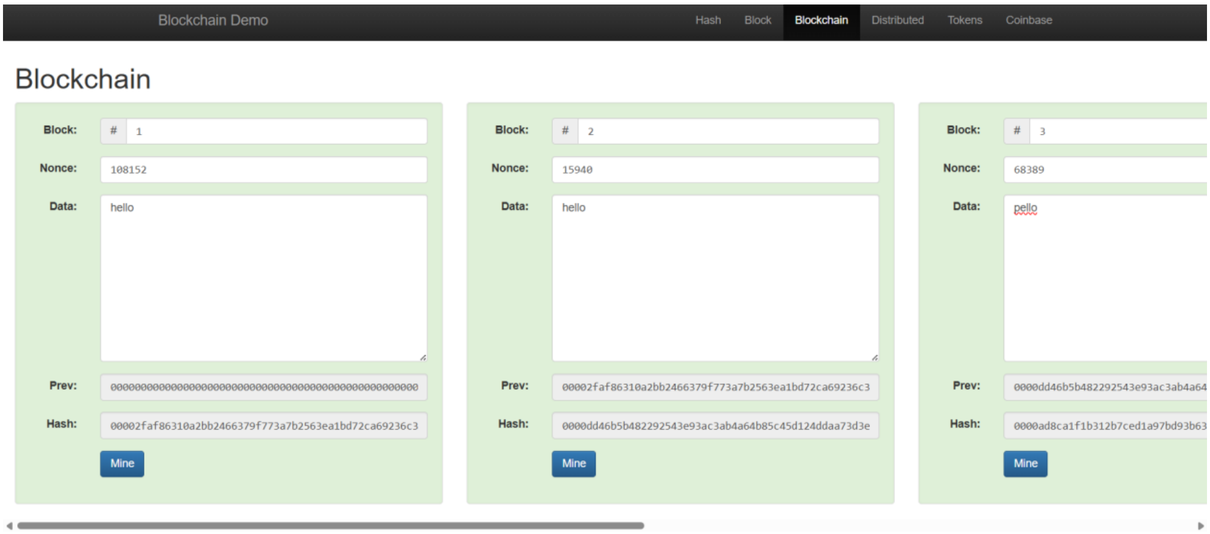
A **blockchain** is a **decentralized and distributed digital ledger** that securely records data in a sequence of blocks, with each block containing information such as transactions, a timestamp, a nonce (in case of proof-of-work), and the cryptographic hash of the previous block. This linking of blocks through hashes ensures that once data is recorded, it becomes **immutable and tamper-resistant**, because altering one block would require changing all subsequent blocks and gaining consensus from the majority of the network.

Blockchain operates without a central authority, using **consensus mechanisms** like Proof-of-Work, Proof-of-Stake, or other protocols to validate and add new blocks. Every participant in the network maintains a copy of the blockchain, providing **transparency** and enabling trustless verification of transactions. Its key characteristics include **security, decentralization, transparency, and immutability**, which make it suitable for applications like cryptocurrencies (e.g., Bitcoin), smart contracts, supply chain tracking, and secure data management.

Output:







Question of Curiosity:

Q.1. Mention features of blockchain to maintain integrity of data

1. **Immutability:** Once data is recorded in a block, it cannot be altered or deleted without modifying all subsequent blocks, which is computationally infeasible.
2. **Hash Linking:** Each block contains the cryptographic hash of the previous block, creating a chain. Any change in a block would change its hash, breaking the chain and revealing tampering.
3. **Decentralization:** Copies of the blockchain are maintained by multiple nodes in the network. Data integrity is preserved even if some nodes fail or act maliciously.
4. **Consensus Mechanisms:** Protocols like Proof-of-Work or Proof-of-Stake ensure that all participants agree on the validity of new blocks before they are added.
5. **Transparency & Auditability:** All transactions are visible to participants and can be independently verified, ensuring traceability and accountability.
6. **Time-stamping:** Each block contains a timestamp, providing chronological order and making it easier to detect fraudulent modifications.

These features together make blockchain **tamper-resistant and reliable** for maintaining secure, trustworthy data.

Conclusion:

Blockchain is a decentralized, tamper-resistant ledger that maintains data integrity through features like immutability, hash linking, decentralization, consensus mechanisms, transparency, and time-stamping. These characteristics ensure that recorded data cannot be altered, is verifiable by all participants, and remains secure and trustworthy across the network.

Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 14/07/2025	Date/Time of Submission : 14/08/2025
Grade :	

EXPERIMENT 2

AIM: Design an Algorithm and implement a program to demonstrate Digital Signature in any coding language of your choice

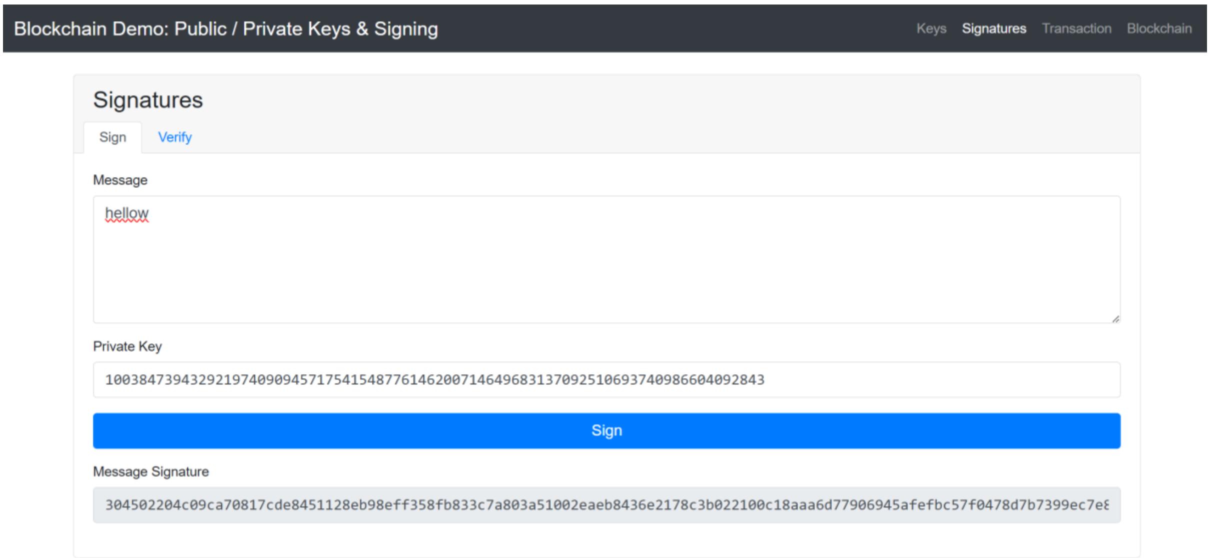
```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization
def generate_keys():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )
    public_key = private_key.public_key()
    return private_key, public_key
def sign_message(private_key, message: bytes) -> bytes:
    signature = private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature
def verify_signature(public_key, message: bytes, signature: bytes) -> bool:
    try:
        public_key.verify(
            signature,
            message,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return True
    except Exception:
```

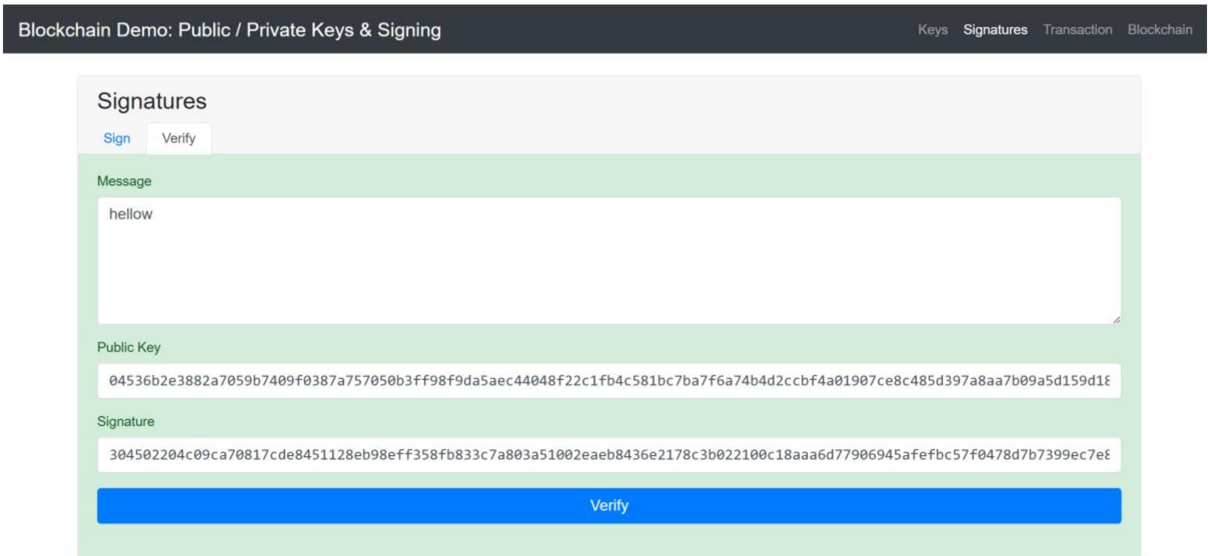
```python
        return False
def main():
    message = b"Hello, this is a confidential message."
    private_key, public_key = generate_keys()
    signature = sign_message(private_key, message)
    print(f"Signature: {signature.hex()}")
    is_valid = verify_signature(public_key, message, signature)
    print(f"Signature valid? {is_valid}")
    tampered_message = b"Hello, this is a tampered message."
    is_valid_tampered = verify_signature(public_key, tampered_message, signature)
    print(f"Signature valid on tampered message? {is_valid_tampered}")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
input
aca919939f2e532daea958e38971a0b528aa9858240537cc8bd6489fb4b6e2a
Signature valid? True
Signature valid on tampered message? False


...Program finished with exit code 0
Press ENTER to exit console.
```

a. **Test the following interactive demo of Digital Signatures and explain your observations about the results obtained while Signing and verifying the signatures:** [**https://andersbrownworth.com/blockchain/public-private-keys/signatures**](https://andersbrownworth.com/blockchain/public-private-keys/signatures)





a. **Evaluate the following scenario and demonstrate a detailed solution for the problem using Digital Signature:**
Mr X wants to send the company's tender information to Mr Y, who works in the other branch for confirmation. Both Mr X and Mr Y want confidence that the tender information has not been intercepted by third person Mr Z on route and altered.

Mr. X creates a hash of the tender, encrypts it with his private key to make a digital signature, and sends both the tender and signature to Mr. Y. Mr. Y hashes the received tender and decrypts the signature with Mr. X's public key. If the hashes match, the tender is authentic and unaltered, ensuring no interception or tampering by Mr. Z.

a. **IIT VLABS:** Digital Signature Simulation:
https://cse29-iiith.vlabs.ac.in/exp/digital-signatures/

Digitally sign the plaintext with Hashed RSA.

Plaintext (string):

test [SHA-1]

Hash output(hex):

a94a8fe5ccb19ba61c4c0873d391e987982fbbd3

Input to RSA(hex):

a94a8fe5ccb19ba61c4c0873d391e987982fbbd3 [Apply RSA]

Digital Signature(hex):

06383c6bfcdde89de5fe2f58c27f75b47ec99326e8111adbae16d2cfcb408ae1
73a23c8b2ef2cce589a3b6505ca9e9b7ab2cf4bfb402b8fc79826c499c7fd69c
d49dfa4de7af8a0a2093fdfe6eb451767d2c2210c6dea2b53cb222b9697829de
2ff486a4ee83de5ff149277c90f8885877d7d0c4418af01c447d2a2e20d904e6

Digital Signature(base64):

Bjg8a/zd6J31/i9Ywn91tH7JkyboERrbrhbSz8tAiuFzojyLLvLM5YmjtlBcqem3
qyz0v7QCuPx5gmxJnH/WnNSd+k3nr4oKIJP9/m60UXZ9LCIQxt6itTyyIrlpeCne
L/SGpO6D3l/xSSd8kPiIWHfX0MRBivAcRH0qLiDZBOY=

Status:

Time: 7ms

RSA public key

Public exponent (hex, F4=0x10001):

10001

Modulus (hex):

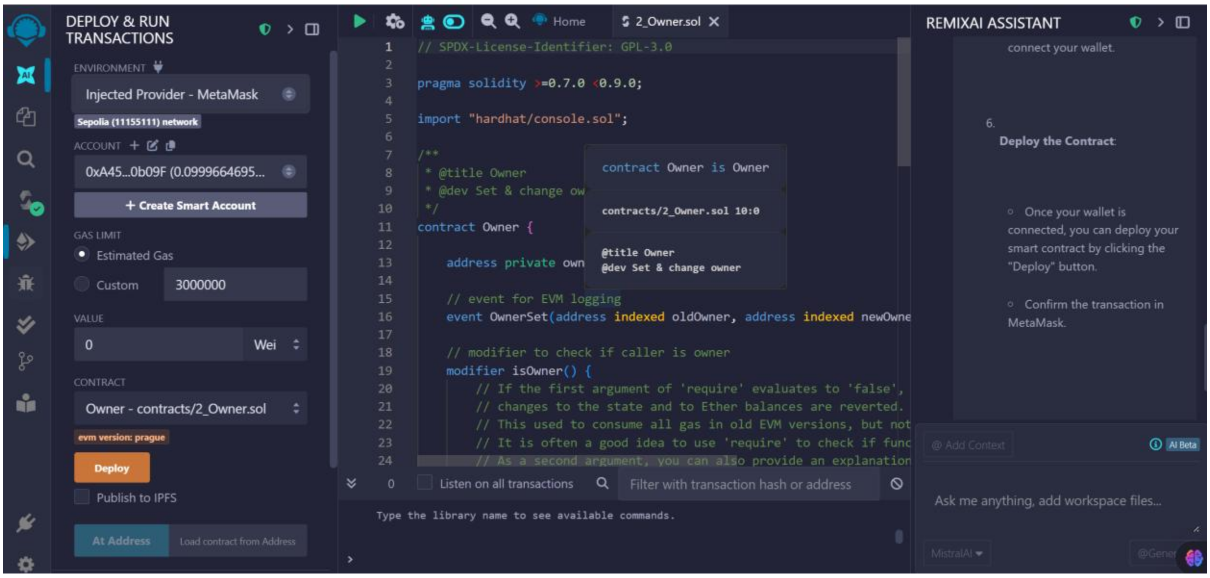
a5261939975948bb7a58dffe5ff54e65f0498f9175f5a09288810b8975871e99
af3b5dd94057b0fc07535f5f97444504fa35169d461d0d30cf0192e307727c06
5168c788771c561a9400fb49175e9e6aa4e23fe11af69e9412dd23b0cb6684c4
c2429bce139e848ab26d08290733 51f4acd36074eafd036a5eb83359d2a698d3

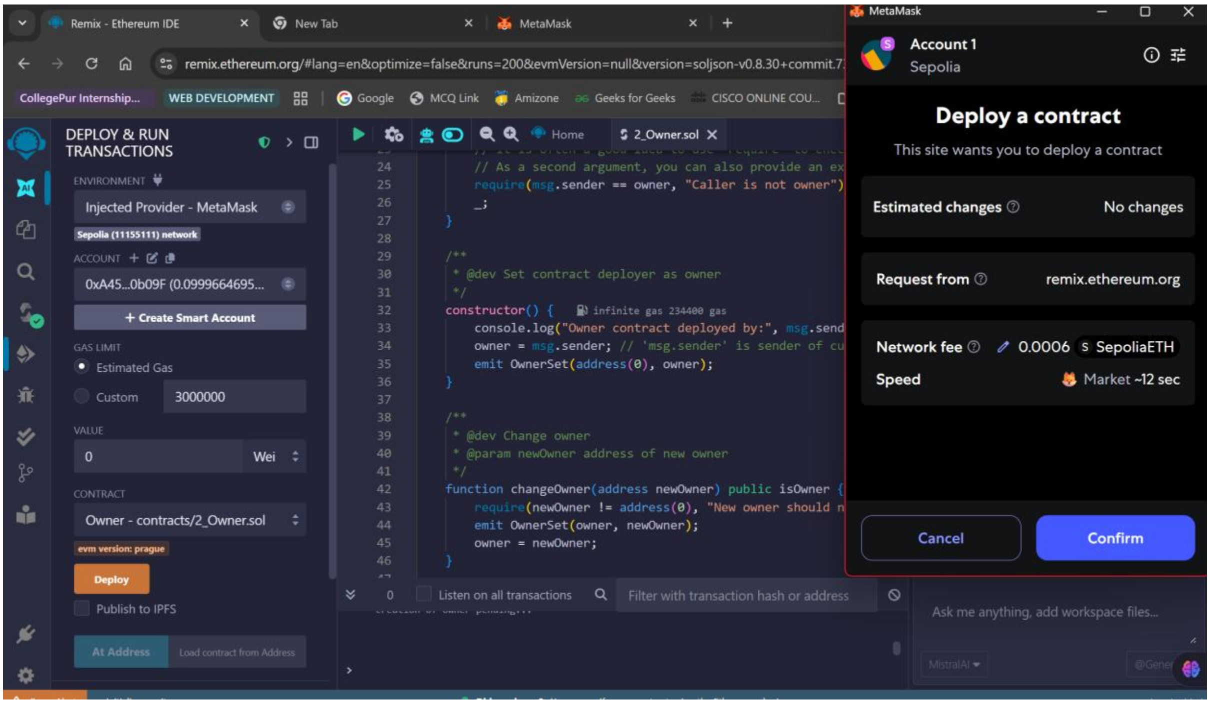
[1024 bit] [1024 bit (e=3)] [512 bit] [512 bit (e=3)]

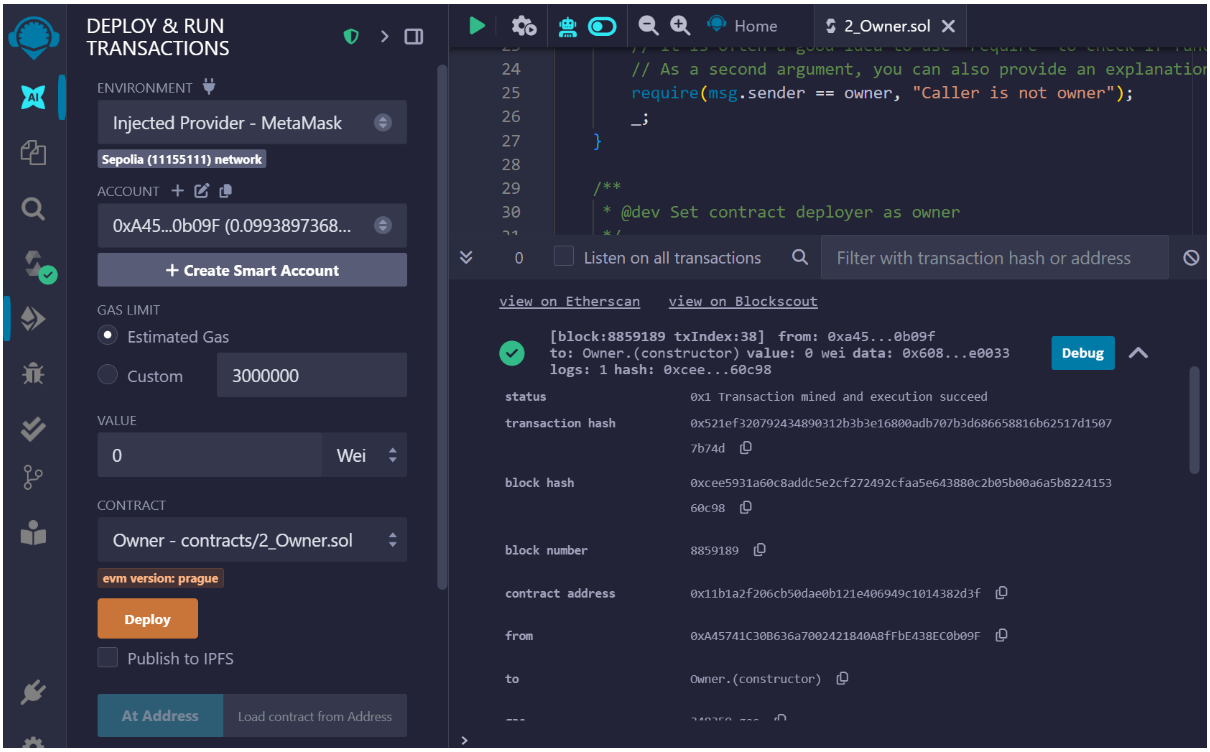
Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 22/07/2025	Date/Time of Submission : 22/07/2025
Grade :	

EXPERIMENT -3

To demonstrate how a decentralized application (DApp) connects with the MetaMask Ethereum wallet from both a user and developer perspective.







status	0x1 Transaction mined and execution succeed
transaction hash	0x521ef320792434890312b3b3e16800adb707b3d686658816b62517d15077b74d
block hash	0xcee5931a60c8addc5e2cf272492cfaa5e643880c2b05b00a6a5b822415360c98
block number	8859189
contract address	0x11b1a2f206cb50dae0b121e406949c1014382d3f
from	0xA45741C30B636a7002421840A8fFbE438EC0b09F
to	Owner.(constructor)
gas	348359 gas
transaction cost	344458 gas
input	0x608...e0033
decoded input	{}

decoded output	-
logs	[{ "from": "0x11b1a2f206cb50dae0b121e406949c1014382d3f", "topic": "0x342827c97908e5e2f71151c08502a66d44b6f758e3ac2f1de95f02eb95f0a735", "event": "OwnerSet", "args": { "0": "0x0000000000000000000000000000000000000000", "1": "0xA45741C30B636a7002421840A8fFbE438EC0b09F" } }]
raw logs	[{ "address": "0x11b1a2f206cb50dae0b121e406949c1014382d3f", "blockHash": "0xcee5931a60c8addc5e2cf272492cfaa5e643880c2b05b00a6a5b822415360c98", "blockNumber": "0x872e35", "data": "0x", "logIndex": "0x53", "removed": false, "topics": ["0x342827c97908e5e2f71151c08502a66d44b6f758e3ac2f1de95f02eb95f0a735", "0x0000000000000000000000000000000000000000000000000000000000000000", "0x000000000000000000000000a45741c30b636a7002421840a8ffbe438ec0b09f"], "transactionHash": "0x521ef320792434890312b3b3e16800adb707b3d686658816b62517d15077b74d", "transactionIndex": "0x26" }]

What is MetaMask and why is it used in DApps?

MetaMask is a browser extension that acts like a digital wallet for cryptocurrencies. It allows users to store, send, and receive Ethereum and other tokens easily. In decentralized applications (DApps), MetaMask is used to connect users' wallets to the app. This connection helps the DApp identify the user and access their digital assets securely. MetaMask acts as a bridge between the user and blockchain networks, making it simple for users to interact with smart contracts and perform transactions without needing to understand complex blockchain details. It provides security by managing private keys locally on the user's device. Overall, MetaMask simplifies blockchain interactions, making DApps more accessible and user-friendly for everyone.

What is the purpose of eth_requestAccounts?

eth_requestAccounts is a command used in web3 applications to ask MetaMask for permission to access the user's Ethereum accounts. When a user visits a DApp and the app wants to interact with their wallet, it calls this method. If the user agrees, MetaMask shares their account addresses with the DApp. This step is essential for the DApp to know which user's account to work with, such as for sending transactions or checking balances. It helps give users control over their privacy by asking for permission before sharing account info. Without this permission, the DApp cannot access user accounts. It is a key part of connecting a user's wallet securely to a DApp, enabling smooth blockchain interactions.

Can MetaMask connect to both testnets and mainnet?

Yes, MetaMask can connect to both testnets and the main Ethereum network. Mainnet is the real, live network where actual cryptocurrencies and assets are traded. Testnets are separate networks used for testing purposes without risking real money. Developers and users can switch between these networks easily in MetaMask. For example, they might use the Ropsten or Rinkeby testnets to test applications before deploying on the mainnet. Switching networks is straightforward, and

MetaMask remembers your selected network. This feature helps users experiment, learn, and develop smart contracts or DApps safely on testnets, and then switch to mainnet when they are ready to go live.

How can a DApp listen to account changes?

A DApp can listen for account changes using event listeners in the browser. When a user switches accounts in MetaMask, the DApp detects this change with a special event called accountsChanged. The DApp registers a handler function that runs whenever this event occurs. This handler updates the app with the new account info, so the user's data stays accurate. This is useful if someone switches to another wallet or a different account within MetaMask. Listening to account changes makes the DApp more dynamic, ensuring it always operates with the current user account, and helps provide a seamless user experience without needing to reload the page manually.

What happens when a user switches the network?

When a user switches the network in MetaMask, such as moving from the Ethereum mainnet to a testnet or another blockchain, the DApp detects this change through a networkChanged event. The DApp then updates its settings to connect to the new network. It might need to reload the page or reinitialize certain functions to work correctly on the new network. Switching networks can affect transactions, balances, and smart contract interactions since each network has different data. Therefore, DApps need to respond to network changes to ensure users see current info and perform actions on the right blockchain. It allows users to switch contexts easily, from testing environments to live networks.

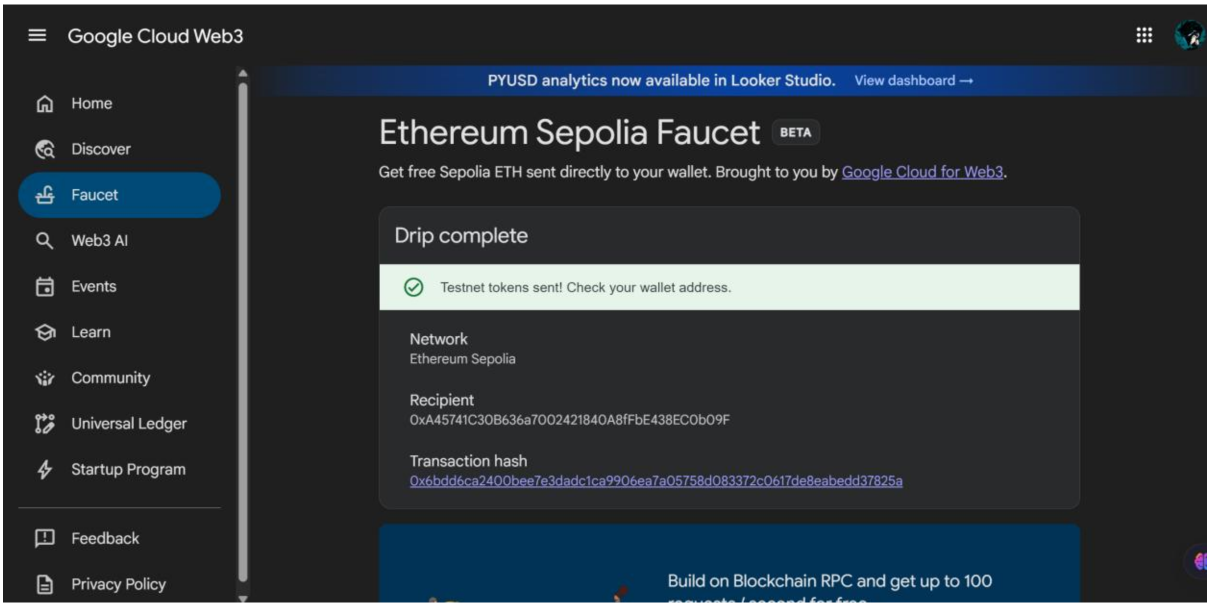
Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 28/07/2025	Date/Time of Submission : 28/07/2025
Grade :	

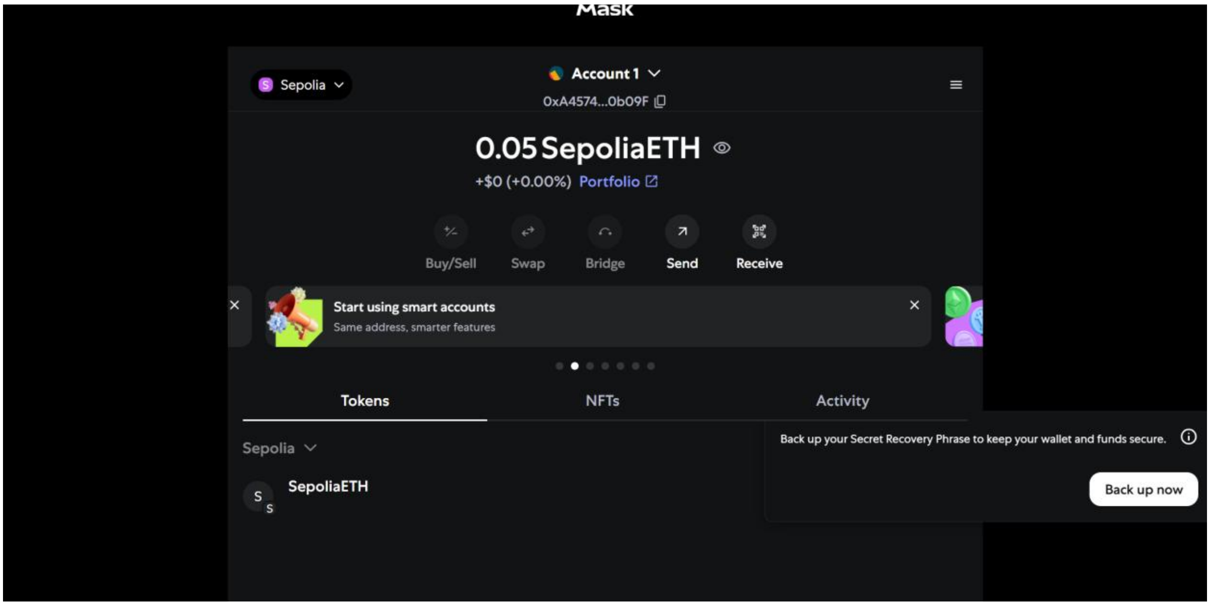
EXPERIMENT 4

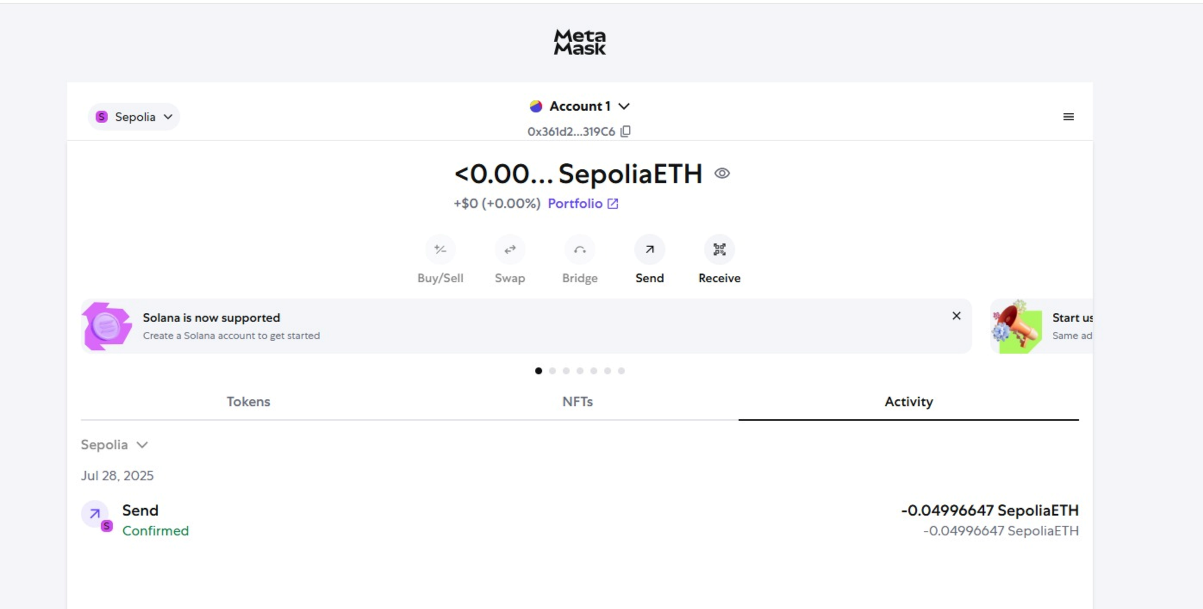
1. Setup Metamask in the System and

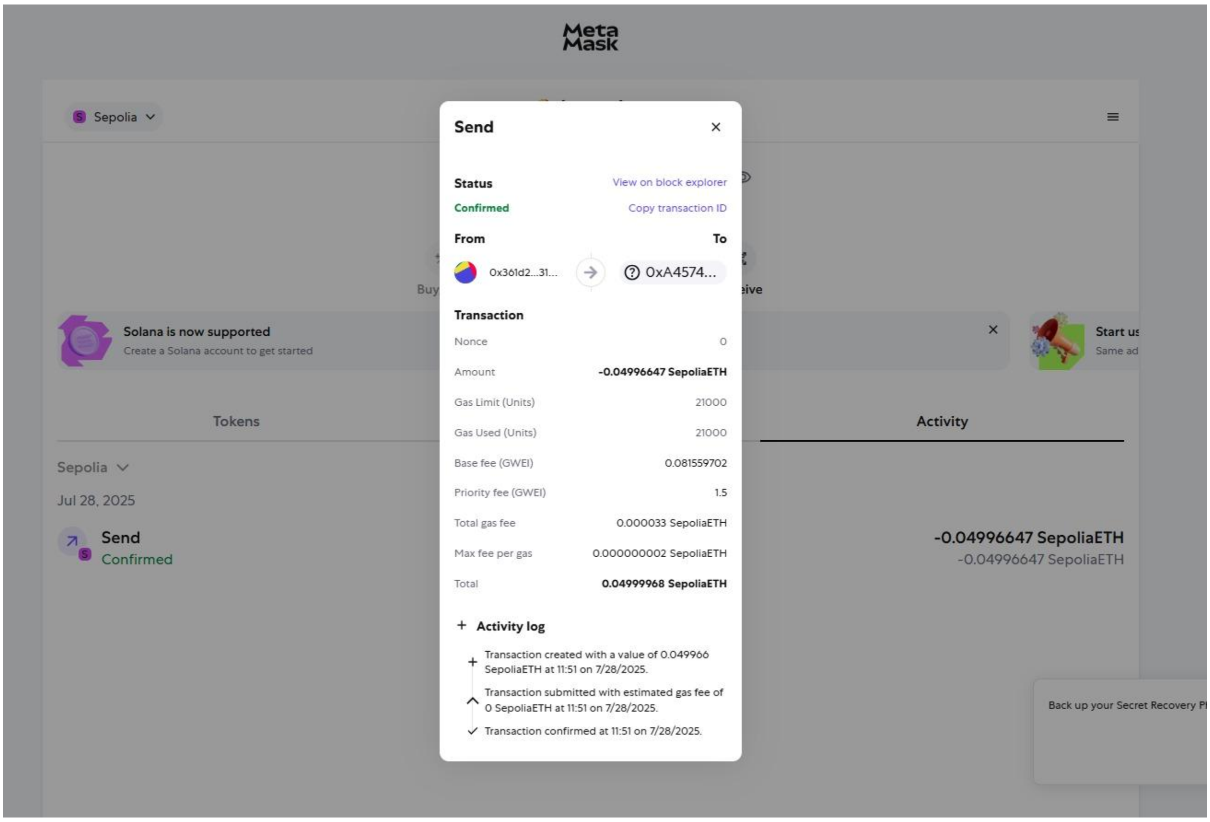
(a) Create a wallet in the Metamask with Test Network

(b) Create multiple accounts in metamask, perform the balance transfer between the accounts, and describe the transaction specifications









Q.1. Why wallets need to be keep secure and what kind of measures does wallets provider to keep it secure use?

Wallets need to be kept secure because they store valuable digital assets like cryptocurrencies, which can be targeted by hackers. If a wallet is not secure, someone could steal the funds or personal information. To protect wallets, providers use several measures. They often include strong passwords and two-factor authentication (2FA), which requires a second step to verify identity. Wallets also use advanced encryption to protect data and transactions. Many providers offer secure backup options, so users don't lose access if they forget their password. Some wallets are hardware-based, meaning they store assets offline, which reduces the risk of hacking. Regular software updates and security patches are also provided to fix vulnerabilities. Overall, these measures make it difficult for unauthorized people to access or steal the digital assets stored in the wallet, ensuring user security and peace of mind.

Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 28/07/2025	Date/Time of Submission : 28/07/2025
Grade :	

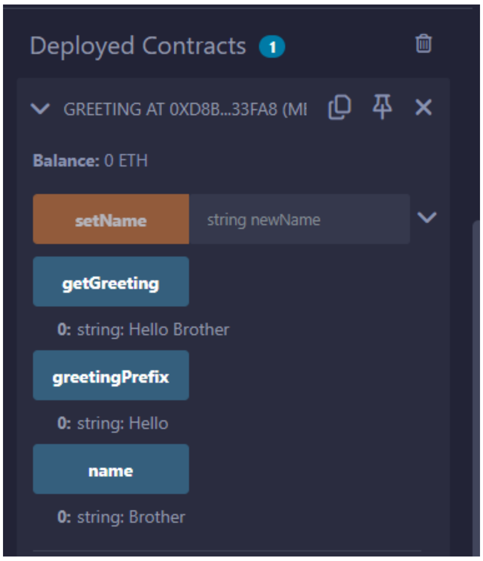
EXPERIMENT 5

Aim: Study and Understand the use of Smart Contracts in Blockchain

Program:

a) **Consider the following link to create a simple smart contract (greeting).**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.11;

contract Greeting {
    string public name;
    string public greetingPrefix = "Hello ";

    constructor(string memory initialName) {     infinite gas 403800 gas
        name = initialName;
    }

    function setName(string memory newName) public {     infinite gas
        name = newName;
    }

    function getGreeting() public view returns (string memory) {     infinite gas
        return string(abi.encodePacked(greetingPrefix, name));
    }
}
```



b) Construct smart contract algorithms for the following scenarios:

a) A transfers property to B for life, and after his death to C and D, equally to be divided between them, or to the survivor of them. C dies during the life of B. D survives B. At B's death the property passes to D.

1. Store lifeTenant = B, remaindermen = [C, D], share = [50%, 50%].
2. Allow authorized reporter to mark addresses as deceased.
3. On receiving event that B is deceased, compute alive remaindermen:
   If both alive → transfer equal shares to C & D.
   If one deceased and the other alive → transfer whole remainder to survivor.
   If both deceased → transfer to fallback beneficiary (if specified) or hold in contract.
4. Execute on-chain transfers of property tokens (or update ownership registry).
5. Emit events for traceability.

b) Mr.X buys a house and is granted a mortgage loan from Bank A. Bank A puts a lien on the property's title and becomes the first lien. After a few months, Mr.X secures another loan using the same property as collateral, but this time from Bank B. Now Mr.X's house has two liens attached to it.Mr.X's ends up defaulting on both mortgages and the banks decide to sell the house. When the house sells at a foreclosure auction, Bank A is the first to recoup its investment. Bank B, as the second lien holder, gets whatever is left after Bank A is paid.

1. When a new mortgage is taken, call recordLien(lender, amount); assign priority based on next sequence number or timestamp.
2. On foreclosure sale (auction result), contract receives salePrice.
3. Deduct auction/administrative fees and any senior lien remedies required by law (we model fees as first fixed amount).
4. Sort liens by priority ascending (1 = first lien). For each lien:
   Pay min(remainingProceeds, lien.balance).
   Reduce remaining proceeds; if remaining becomes 0 break.
5. Emit events and record any unpaid deficiency amounts owed to lienholders.
6. Transfer residual funds to owner or other entitled party (if applicable).

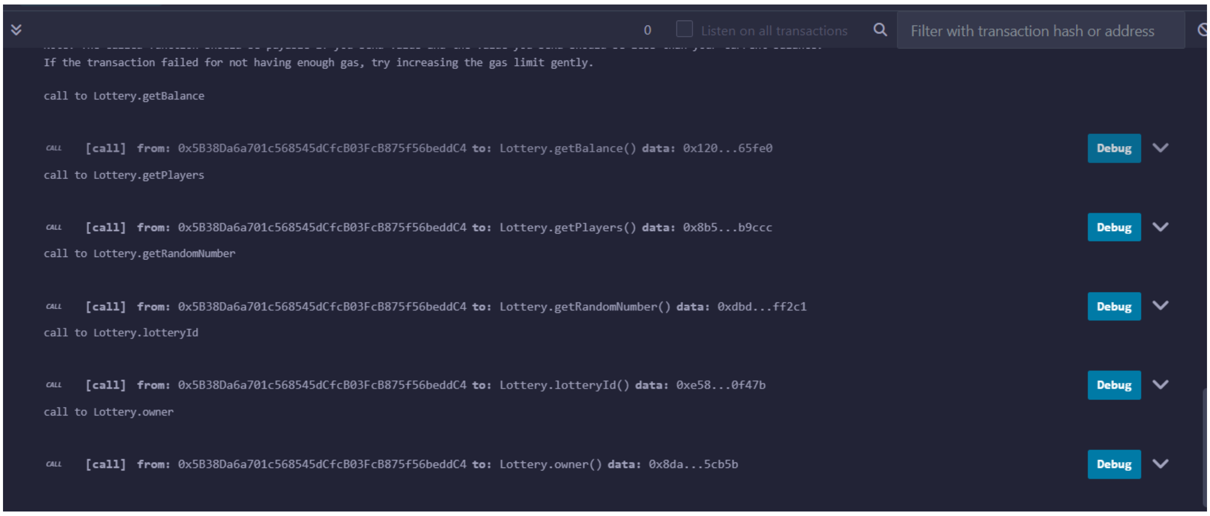
c) Create smart contract on Lottery system

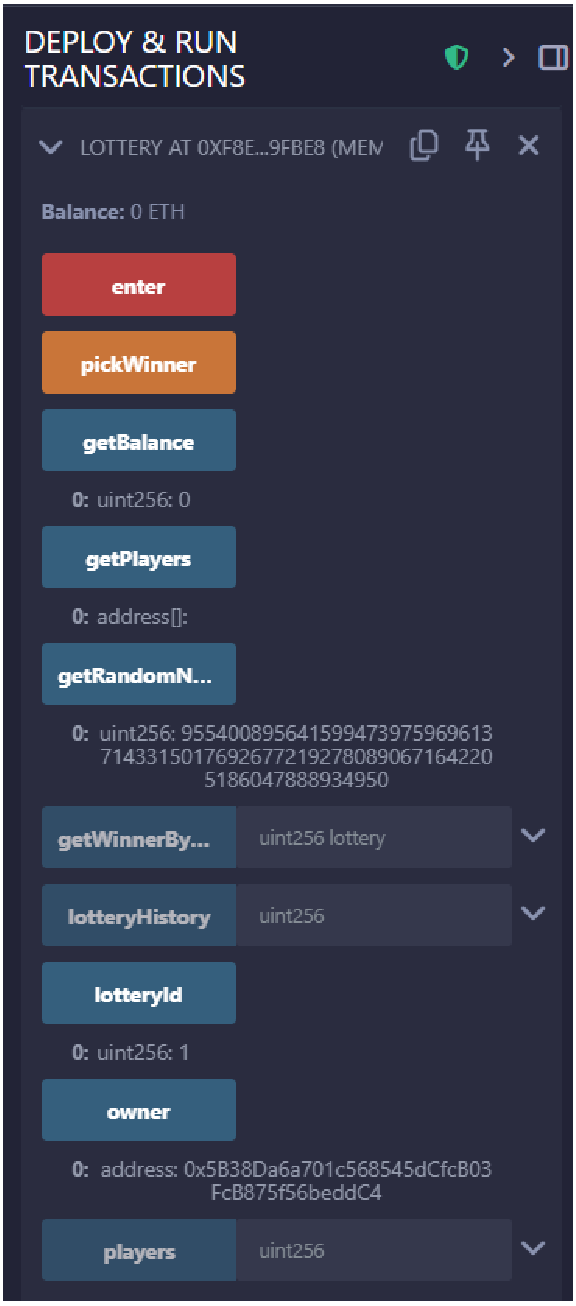
Compile | Hello.sol | Lottery.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.11;

contract Lottery {
    address public owner;
    address payable[] public players;
    uint public lotteryId;
    mapping (uint => address payable) public lotteryHistory;

    constructor() {    602563 gas 555600 gas
        owner = msg.sender;
        lotteryId = 1;
    }

    function getWinnerByLottery(uint lottery) public view returns (address payable) {    2875 gas
        return lotteryHistory[lottery];
    }

    function getBalance() public view returns (uint) {    335 gas
        return address(this).balance;
    }

    function getPlayers() public view returns (address payable[] memory) {    infinite gas
        return players;
    }
```

Compile | Hello.sol | Lottery.sol

```

    function enter() public payable {    48782 gas
        require(msg.value > .01 ether);

        // address of player entering lottery
        players.push(payable(msg.sender));
    }

    function getRandomNumber() public view returns (uint) {    infinite gas
        return uint(keccak256(abi.encodePacked(owner, block.timestamp)));
    }

    function pickWinner() public onlyowner {    infinite gas
        uint index = getRandomNumber() % players.length;
        players[index].transfer(address(this).balance);

        lotteryHistory[lotteryId] = players[index];
        lotteryId++;


        // reset the state of the contract
        players = new address payable[](0);
    }

    modifier onlyowner() {
      require(msg.sender == owner);
      _;
    }
```


0
Listen on all transactions
Filter with transaction hash or address
If the transaction failed for not having enough gas, try increasing the gas limit gently.
call to Lottery.getBalance
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Lottery.getBalance() data: 0x120...65fe0
Debug
call to Lottery.getPlayers
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Lottery.getPlayers() data: 0x8b5...b9ccc
Debug
call to Lottery.getRandomNumber
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Lottery.getRandomNumber() data: 0xdbd...ff2c1
Debug
call to Lottery.lotteryId
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Lottery.lotteryId() data: 0xe58...0f47b
Debug
call to Lottery.owner
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Lottery.owner() data: 0x8da...5cb5b
Debug


DEPLOY & RUN TRANSACTIONS
LOTTERY AT 0XF8E...9FBE8 (MEM
Balance: 0 ETH
enter
pickWinner
getBalance
0: uint256: 0
getPlayers
0: address[]:
getRandomN...
0: uint256: 95540089564159947397596961371433150176926772192780890671642205186047888934950
getWinnerBy...
uint256 lottery
lotteryHistory
uint256
lotteryId
0: uint256: 1
owner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
players
uint256

Conclusion:

The Lottery smart contract is a practical example of implementing decentralized applications on Ethereum using Solidity.It allows participants to enter by sending Ether, with the contract securely managing entries and selecting a winner.This system removes the need for intermediaries, ensuring transparency and trust through blockchain execution.

However, for real-world deployment, enhancements like secure randomness and anti-manipulation measures are essential.

Question of Curiosity:

Q1. Distinguish between Solidity and Chaincode

Aspect	Solidity	Chaincode
Platform	Used for Ethereum and other EVM-compatible blockchains.	Used for Hyperledger Fabric.
Language Type	High-level, contract-oriented programming language similar to JavaScript.	Can be written in Go, Java, or JavaScript.
Purpose	Develops and implements smart contracts on Ethereum.	Implements business logic (smart contracts) for Hyperledger Fabric.
Execution	Runs on the Ethereum Virtual Machine (EVM).	Runs in a Docker container within Fabric peer nodes.
Consensus Mechanism Dependency	Works with Ethereum's consensus mechanisms like Proof of Work (PoW) or Proof of Stake (PoS).	Works with Hyperledger's pluggable consensus (e.g., Raft, Kafka).
Use Cases	Public blockchain applications such as DeFi, NFTs, DAOs.	Enterprise/private blockchain applications requiring permissioned access and privacy.

Q2. Differentiate between Bitcoin and Ethereum smart contracts in terms of coding and deployment

Aspect	**Bitcoin Smart Contracts**	**Ethereum Smart Contracts**
Programming Language	Written in **Bitcoin Script**, a simple, stack-based language with limited functionality.	Written in **Solidity** (or Vyper), a high-level, feature-rich programming language.

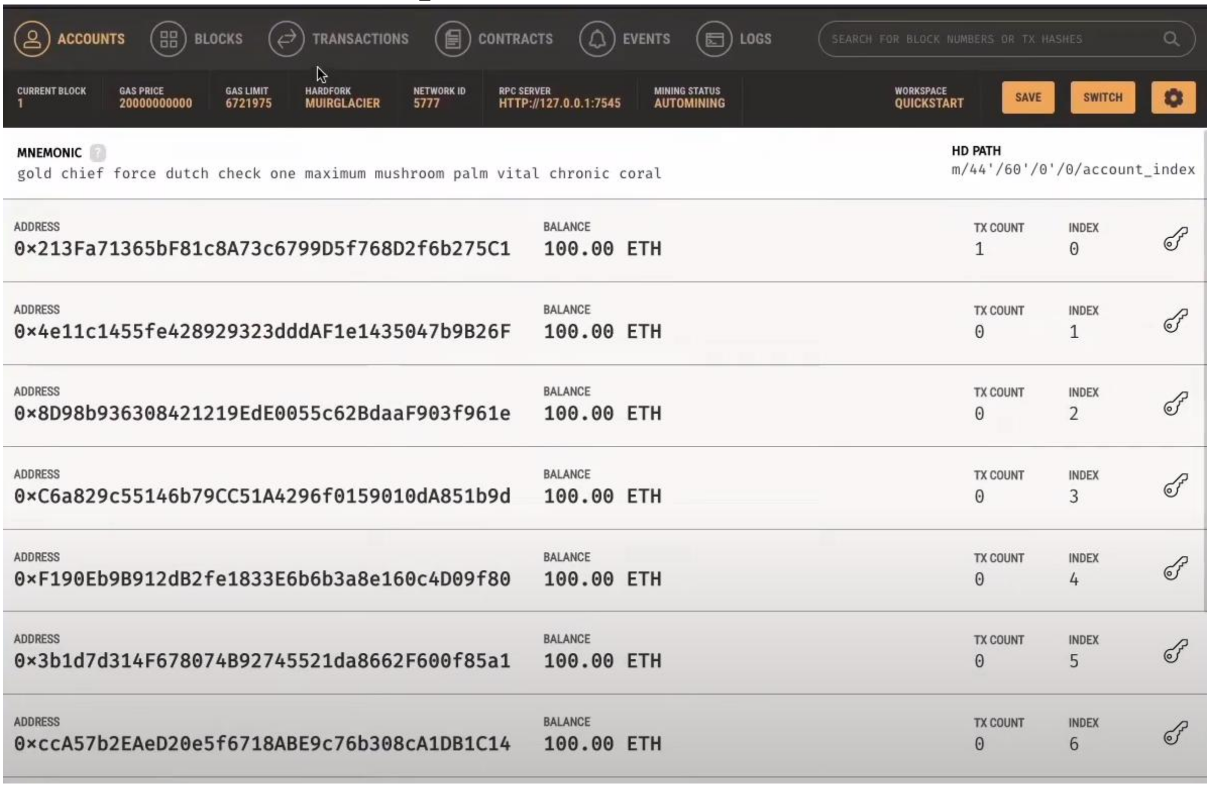
Aspect	Bitcoin Smart Contracts	Ethereum Smart Contracts
Turing Completeness	**Not Turing-complete** – no loops or complex logic to avoid high computational costs and vulnerabilities.	**Turing-complete** – supports loops, complex logic, and stateful computations.
Complexity	Supports basic conditional transactions like multi-signature wallets, time-locks, and escrow.	Supports complex applications like DeFi platforms, DAOs, NFTs, and oracles.
Deployment Process	Embedded directly in Bitcoin transactions; executed by Bitcoin nodes during transaction validation.	Deployed as bytecode to the Ethereum blockchain via transactions; executed on the Ethereum Virtual Machine.
Primary Use	Securing peer-to-peer transactions with limited programmability.	Powering decentralized applications (dApps) with extensive functionality and interoperability.

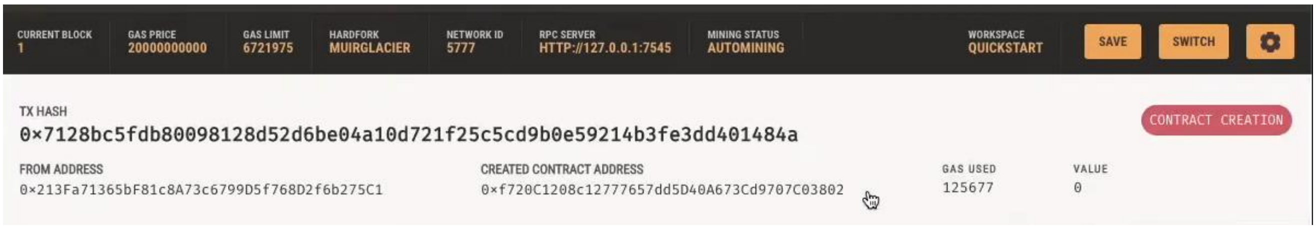
Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 11/08/2025	Date/Time of Submission : 11/08/2025
Grade :	

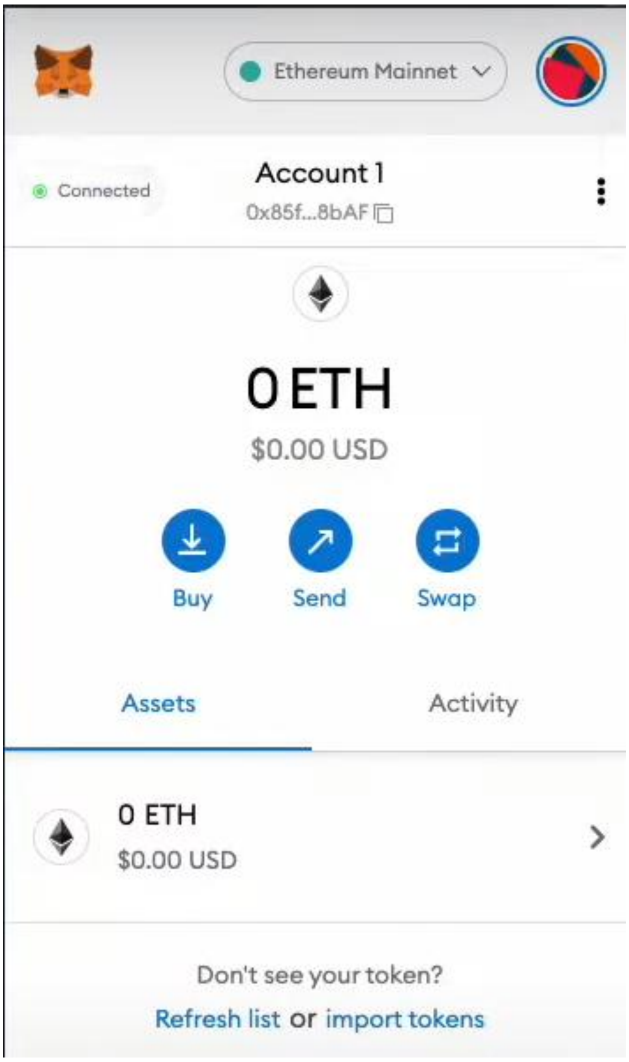
EXPERIMENT 6

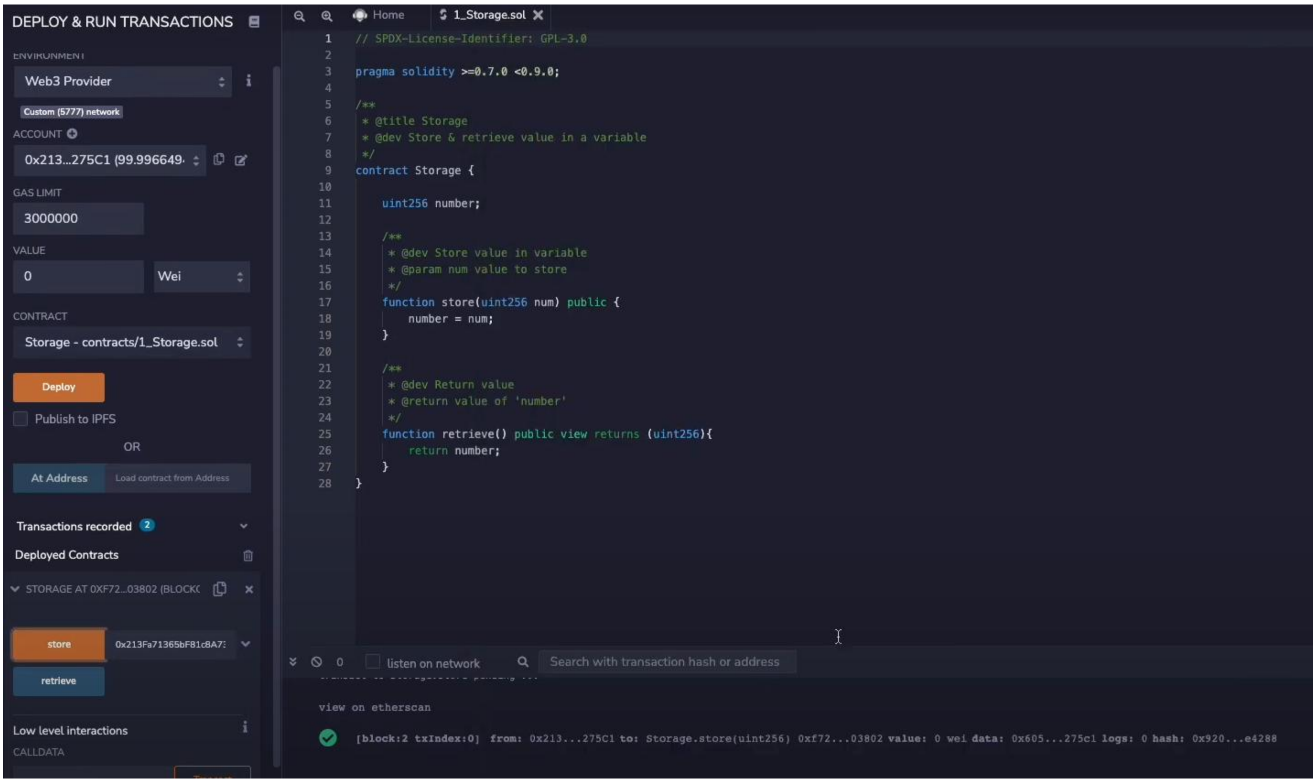
Aim: To install and configure **Ganache Ethereum tool** and interface it with **MetaMask** and **Remix IDE** for smart contract deployment and testing.

B.1 Task with their output:






Ethereum Mainnet
Connected
Account 1
0x85f...8bAF
0 ETH
$0.00 USD
Buy
Send
Swap
Assets
Activity
0 ETH
$0.00 USD
Don't see your token?
Refresh list or import tokens


DEPLOY & RUN TRANSACTIONS
Home
1_Storage.sol
Web3 Provider
Custom (5777) network
ACCOUNT
0x213...275C1 (99.996649.
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
Storage - contracts/1_Storage.sol
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 2
Deployed Contracts
STORAGE AT 0XF72...03802 (BLOCKC
store
retrieve
Low level interactions
CALLDATA
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
/**
* @title Storage
* @dev Store & retrieve value in a variable
*/
contract Storage {
uint256 number;
/**
* @dev Store value in variable
* @param num value to store
*/
function store(uint256 num) public {
number = num;
}
/**
* @dev Return value
* @return value of 'number'
*/
function retrieve() public view returns (uint256){
return number;
}
}
listen on network
Search with transaction hash or address
view on etherscan

Code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 */
contract Storage {
    uint256 number;
    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }
    /**
     * @dev Return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

B.2 Conclusion:

Learned how **Ganache, MetaMask, and Remix** integrate to provide a complete environment for developing and testing smart contracts. Ganache offers a local blockchain for safe testing, MetaMask acts as the wallet and bridge for accounts, and Remix provides the IDE for coding and deploying contracts. These tools help students understand the end-to-end smart contract development lifecycle in a controlled environment before moving to real networks.

B.3 Question of Curiosity:

1. Difference between deploying a smart contract on a local blockchain (Ganache) and a public testnet (Goerli, Sepolia):
Deploying on **Ganache** is fast, free, and local, allowing unlimited test transactions without network delays. However, it only simulates the Ethereum blockchain and is not publicly accessible. Deploying on a **public testnet** like Goerli or Sepolia provides a real-world environment with actual network consensus, miner/validator delays, and faucet-based test ETH. This makes testnets more realistic but also slower and sometimes less predictable than Ganache.

2. Why does MetaMask require a private key to import Ganache accounts?
MetaMask is a wallet that manages blockchain accounts, and every Ethereum account is controlled by a **private key**. When using Ganache, accounts are pre-generated locally along with their private keys. To access these accounts in MetaMask, the private key must be imported so MetaMask can sign transactions on behalf of the account. Without the private key, MetaMask cannot verify ownership or authorize transactions from that Ganache account.

Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 18/08/2025	Date/Time of Submission : 18/08/2025
Grade :	

EXPERIMENT 7

Aim: Build Blockchain by using python

Code:

```
import hashlib

import time

import json

class Block:

    def __init__(self, index, timestamp, data, previous_hash, nonce=0):

        self.index = index

        self.timestamp = timestamp

        self.data = data

        self.previous_hash = previous_hash

        self.nonce = nonce

        self.hash = self.calculate_hash()

    def calculate_hash(self):

        block_string = json.dumps({

            "index": self.index,

            "timestamp": self.timestamp,

            "data": self.data,

            "previous_hash": self.previous_hash,

            "nonce": self.nonce

        }, sort_keys=True).encode()

        return hashlib.sha256(block_string).hexdigest()

    def mine_block(self, difficulty):

        target = '0' * difficulty

        while self.hash[:difficulty] != target:
```

```
            self.nonce += 1
            self.hash = self.calculate_hash()
        print(f"Block {self.index} mined: {self.hash}")
class Blockchain:
    def __init__(self, difficulty=2):
        self.chain = [self.create_genesis_block()]
        self.difficulty = difficulty
    def create_genesis_block(self):
        return Block(0, time.time(), "Genesis Block", "0")
    def get_latest_block(self):
        return self.chain[-1]
    def add_block(self, new_block):
        new_block.previous_hash = self.get_latest_block().hash
        new_block.mine_block(self.difficulty)
        self.chain.append(new_block)
    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i-1]
            if current.hash != current.calculate_hash():
                print("Current hash invalid")
                return False
            if current.previous_hash != previous.hash:
                print("Previous hash mismatch")
                return False
        return True
```

```
if __name__ == "__main__":

    my_blockchain = Blockchain(difficulty=3)

    print("Mining block 1...")

    my_blockchain.add_block(Block(1, time.time(), {"amount": 10}, ""))

    print("Mining block 2...")

    my_blockchain.add_block(Block(2, time.time(), {"amount": 20}, ""))

    print("\nBlockchain valid?", my_blockchain.is_chain_valid())

    for block in my_blockchain.chain:

        print(f"Index: {block.index}, Hash: {block.hash}, Data: {block.data}")
```

Output:

```
Mining block 1...
Block 1 mined: 00080a8116fe73f76bdf5177bc16ca86632cfd78f2308fb918e136d4c65a0dec
Mining block 2...
Block 2 mined: 000f08bd97c85e12fc686fe00466caf833229954a42704f538bc5db7e48d444f

Blockchain valid? True
Index: 0, Hash: 6b6e948dd06326b52b1ef6eb85474e9ba167e7e4384254f6778a800a714725c1, Data: Genesis Block
Index: 1, Hash: 00080a8116fe73f76bdf5177bc16ca86632cfd78f2308fb918e136d4c65a0dec, Data: {'amount': 10}
Index: 2, Hash: 000f08bd97c85e12fc686fe00466caf833229954a42704f538bc5db7e48d444f, Data: {'amount': 20}
```

Question of Curiosity:

Q.1. Explain functionality of blockchain-

A blockchain is a **distributed ledger** that records data in a series of **blocks**, each containing a set of transactions or information. Every block is **linked to the previous block** using a cryptographic hash, ensuring the integrity of the chain. When a new block is added, participants in the network validate it using **consensus mechanisms** like Proof-of-Work or Proof-of-Stake. This makes the data **tamper-resistant**, as altering any block would require changing all subsequent blocks and gaining network approval. Blockchain also allows **decentralized record-keeping**, meaning no single authority controls the data, and all participants can verify the chain independently.

Conclusion:

The blockchain experiment was performed successfully, demonstrating how blocks can be linked using hashes and secured with proof-of-work. It showed how data integrity is maintained across the chain and how new blocks are added and validated, illustrating the core principles of blockchain technology.

Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-X	Batch : 2022-2026
Date of Experiment : 18/08/2025	Date/Time of Submission : 25/08/2025
Grade :	

EXPERIMENT 8

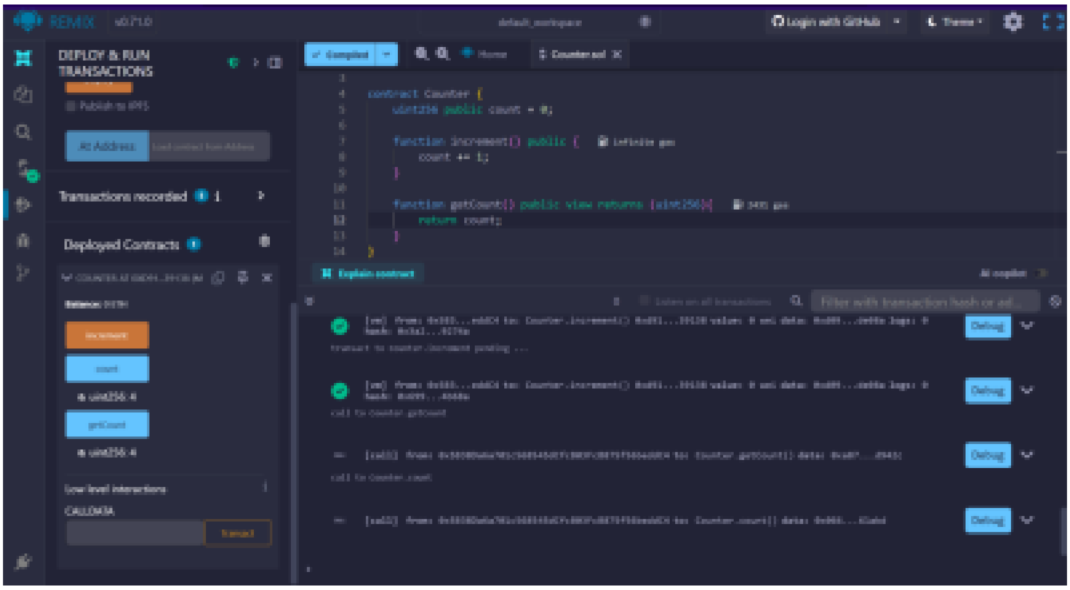
Aim: Implementing Smart Contract.

Theory: A smart contract is a self-executing contract with the terms of the agreement directly written into code. This counter contract demonstrates how a smart contract can store and modify values on the blockchain, allowing decentralized and transparent interactions. Using Ethereum and Solidity, smart contracts can execute logic autonomously.

Procedure:

1. Write the Solidity code for a simple counter contract in Remix IDE.
2. Define a counter variable to store the count value.
3. Implement a function to increment the counter and a function to retrieve the current value.
4. Compile the smart contract in Remix.
5. Deploy the contract to a test network (e.g., Rinkeby).
6. Interact with the contract by calling the increment and retrieve functions.
7. Monitor transaction status and confirm the counter increments on the blockchain.

Observations:



Result:

The contract successfully allows interaction with the counter on the test network. Each increment function call increases the stored value, demonstrating the contract's functionality and interaction with the blockchain.

Roll No. : A023166922038	Name: ARYAN WALIA
Class : 7CSE-IOT-Y	Batch : 2022-2026
Date of Experiment : 22/09/2025	Date/Time of Submission : 22/09/2025
Grade :	