EXPERIMRNT-01

AIM: Develop a Java/Python/Go program to create Elliptic curve public and private keys and demonstrate working of hash functions like SHA256 and ECC digital signatures.

DESCRIPTION:

1. Elliptic Curve Cryptography (ECC)

ECC is a modern cryptographic technique based on elliptic curve theory that provides the same level of security as traditional cryptography (e.g., RSA) but with much smaller keys. This leads to faster computation, reduced power consumption, and lower storage requirements, making it ideal for systems with limited resources like IoT devices or mobile applications.

- **Private Key**: A randomly generated secret number used for signing and generating public keys.
- **Public Key**: Derived from the private key and used for verification.
- **Curve**: The mathematical curve used to generate the keys (e.g., secp256r1 is a commonly used curve in ECC).

2. Hash Functions (SHA-256)

A hash function is a one-way cryptographic function that converts input data (a message) into a fixed-length output (the hash). It is deterministic and produces the same hash for identical inputs.

- **SHA-256**: A cryptographic hash function from the SHA-2 family. It generates a 256-bit (32-byte) hash that is:
  - **Irreversible**: Cannot retrieve the original data from the hash.
  - **Unique**: A small change in the input drastically changes the output (avalanche effect).
  - **Fixed-Length**: Always outputs 256 bits, irrespective of input size.

3. ECC Digital Signatures

A digital signature is a cryptographic technique to ensure:

- **Authentication**: Verifies the identity of the sender.
- **Integrity**: Ensures the message was not altered.
- **Non-Repudiation**: Prevents the sender from denying sending the message.

The ECC digital signature algorithm involves:

1. **Signing**:
   - The private key is used to sign a message hash (e.g., SHA-256 hash).
   - A signature is generated as a pair of numbers (r, s) derived from the elliptic curve operations.
2. **Verification**:
   - The public key is used to validate the signature against the original message hash.

Source code:

```
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import hashes
from cryptography.exceptions import InvalidSignature
private_key = ec.generate_private_key(ec.SECP256R1())
public_key = private_key.public_key()
message = input("Enter the message you want to sign: ").encode()
digest = hashes.Hash(hashes.SHA256())
digest.update(message)
hash_value = digest.finalize()
print("\nSHA-256 Hash:", hash_value.hex())
signature = private_key.sign(hash_value, ec.ECDSA(hashes.SHA256()))
print("\nDigital Signature:", signature.hex())
try:
    public_key.verify(signature, hash_value, ec.ECDSA(hashes.SHA256()))
    print("\nSignature is valid!")
except InvalidSignature:
    print("\nInvalid signature!")
```

Output :

```
Enter the message you want to sign: i am sathwika Akkala

SHA-256 Hash: 12d03185fabab256d0545384d72fecebb8866e08ce35f3dfaaa81bc553d46e66

Digital Signature: 3044022063bdd9cc442f80b3feda35dd0756b807e56dd1714da048ce6e5302c2
6e415a2a0220108bb156c92e41148a5b12c47adbc9fa63fdc3161e0d7ee12a95b3b64f135727

Signature is valid!
```

EXPERIMRNT-02

AIM: Setup a Bitcoin wallet like Electrum and demonstrate sending and receiving Bitcoins on a testnet. Use Blockchain explorer to observe the transaction details

DESCRIPTION:

Electrum is one of the most popular and lightweight Bitcoin wallets available today. It is designed to provide users with a secure, efficient, and easy way to store, send, and receive Bitcoin. Below is a detailed description of Electrum, its features, and its advantages:

What is Electrum?

Electrum is an open-source Bitcoin wallet that was first released in 2011. It is focused on simplicity and speed, making it an excellent choice for both beginners and advanced Bitcoin users. Electrum uses a decentralized server network to access the Bitcoin blockchain, ensuring reliability and fast performance.

Key Features of Electrum

1. **Lightweight**:
   - Electrum does not download the entire Bitcoin blockchain. Instead, it connects to decentralized servers that provide blockchain data, saving time and storage space.
2. **Secure**:
   - Electrum is highly secure and provides features like two-factor authentication (2FA), multi-signature wallets, and hardware wallet support.
   - Your private keys are encrypted and never leave your computer.
3. **Open Source**:
   - The wallet's source code is publicly available, allowing anyone to audit or contribute to its development.
4. **Cross-Platform**:
   - Electrum is available for Windows, macOS, Linux, and Android.
5. **Customizable Fees**:
   - Users can adjust transaction fees based on urgency. Lower fees are suitable for slower confirmations, while higher fees ensure faster processing.

Procedure:

Step 1: Install Electrum

Electrum is a lightweight Bitcoin wallet. To use the testnet version, you'll need to install Electrum and configure it for testnet.

1. **Download Electrum**:
   - Visit the official Electrum website.
   - Download the latest version for your operating system (Windows, macOS, or Linux).
2. **Run Electrum in Testnet Mode**:
   - On Windows:
     - Open the **Command Prompt**, navigate to the directory where you installed Electrum, and run:

electrum --testnet

Step 2: Set Up a Testnet Wallet

1. **Create a New Wallet**:
   - Choose **"Create a new wallet"** and select **Standard Wallet**.
   - Write down the recovery seed phrase (important for backup).
   - Set a strong password for the wallet.
2. **Switch to Testnet**:

Step 3: Get Testnet Bitcoins

1. **Find a Testnet Faucet**:
   - Use a faucet to get free testnet BTC. Example:
     - https://bitcoinfaucet.uo1.net/
   - Enter your testnet Bitcoin address (you can copy it from Electrum under Receive) to request testnet BTC.
2. **Check Your Balance**:
   - Once the faucet sends the coins, you should see them in your wallet after a few minutes.

Step 4: Send Testnet Bitcoin

1. **Get a Receiver Address**:
   - Use another testnet wallet (or the Receive tab in Electrum on another device) to generate a receiving address.
2. **Send Transaction**:
   - Go to the Send tab in Electrum.
   - Enter the receiver's testnet Bitcoin address and the amount to send.
   - Set an appropriate transaction fee (Electrum will recommend one).
   - Click **Send** and confirm the transaction.
3. **Observe Transaction ID**:
   - Electrum will display the transaction ID after sending. Copy it for tracking.

Step 5: Use a Blockchain Explorer

1. **Open a Testnet Blockchain Explorer**:
   - Example: https://blockstream.info/testnet/
2. **Track the Transaction**:
   - Paste the transaction ID into the explorer's search bar to view details like:
     - Sender and receiver addresses.
     - Transaction amount.
     - Transaction fee.
     - Confirmation status.

Step 6: Receive Bitcoin

1. **Generate a Receiving Address**:
   - In Electrum, go to the Receive tab and copy your receiving address.
2. **Send Bitcoin to This Address**:
   - You can use another wallet or the faucet to send Bitcoin to your address.
3. **Check for Incoming Funds**:
   - Your wallet will show the incoming transaction and update the balance once it's confirmed.

Output:

Example Transactions:

Sending:

- **Transaction ID**: 4b1f35...
- **From Address**: tb1q...
- **To Address**: tb1q...
- **Amount**: 0.001 BTC
- **Fee**: 0.0001 BTC

Receiving:

- Check the testnet blockchain explorer to verify that the transaction has reached your address.

EXPERIMRNT-03

AIM: Setup metamask wallet in a web browser and create wallet and user accounts. Demonstrate sending and receiving ethers on a testnet (Sepolia). Use Block explorers like etherscan to observe the transaction details.

DESCRIPTION:

This experiment involves configuring **MetaMask**, a popular Ethereum wallet, to connect with the **Sepolia Testnet**. Sepolia is a test blockchain that mimics Ethereum's main network but uses test ETH for **safe, risk-free transactions**.

The steps include:
1. **Installing MetaMask** in a web browser.
2. **Enabling test networks** in MetaMask settings.
3. **Adding Sepolia Testnet manually** if not listed.
4. **Obtaining free SepoliaETH** from a faucet.
5. **Sending and receiving ETH transactions**.
6. **Tracking transactions using Sepolia Etherscan**.

This process helps in **testing smart contracts, dApps, and Ethereum transactions** without real financial risks.

Procedure:

Step 1: Install MetaMask

1. Open **Google Chrome** (or any Chromium-based browser like Brave, Edge).
2. Go to the official **MetaMask website**: https://metamask.io/
3. Click **Download** and install the **MetaMask extension** for your browser.
4. After installation, click on the **MetaMask icon** in the browser toolbar.

Step 2: Create a MetaMask Wallet

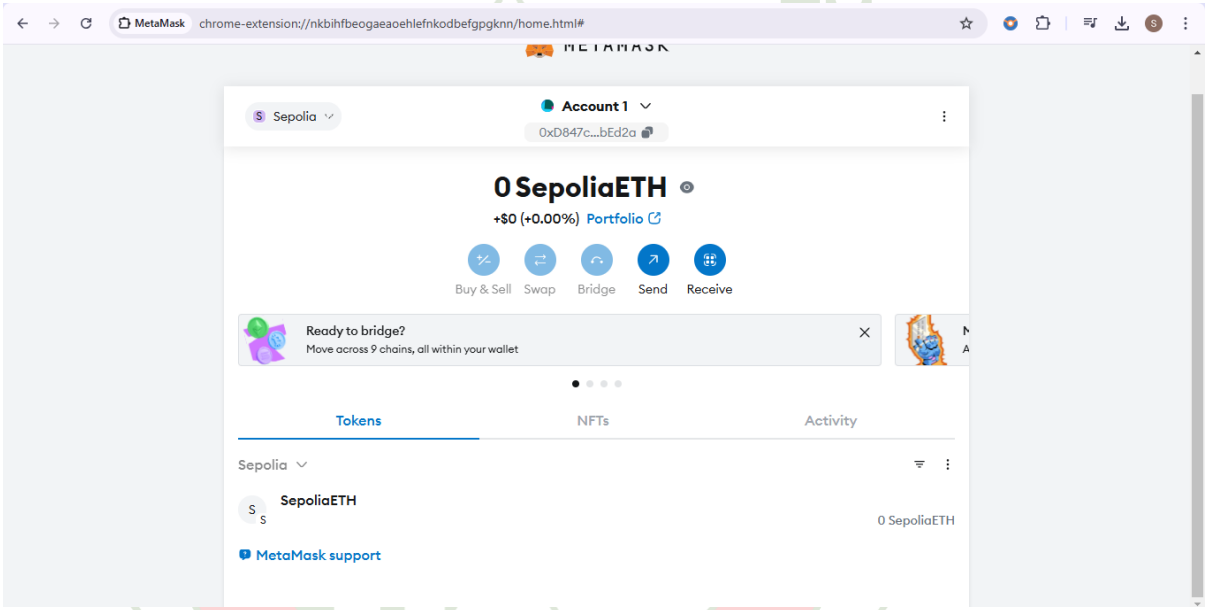
1. Click **Get Started** → Choose **Create a Wallet**.
2. Set a **strong password** and click **Create**.
3. **Backup the Secret Recovery Phrase (12 words)** securely (DO NOT share it).
4. Confirm the recovery phrase to finish wallet creation.

Step 3: Add Sepolia Testnet to MetaMask

1. Open MetaMask and click on the **Network Selector** (top-right corner).
2. Click **Show/Hide Test Networks** → Enable **Sepolia Test Network**.
3. Select **Sepolia** from the network list.

-> If Sepolia is not listed, add it manually:

- **Network Name**: Sepolia Testnet
- **RPC URL**: https://sepolia.infura.io/v3/YOUR_INFURA_PROJECT_ID
- **Chain ID**: 11155111
- **Currency Symbol**: ETH
- **Block Explorer**: https://sepolia.etherscan.io/



Step 4: Get Free Sepolia Test ETH

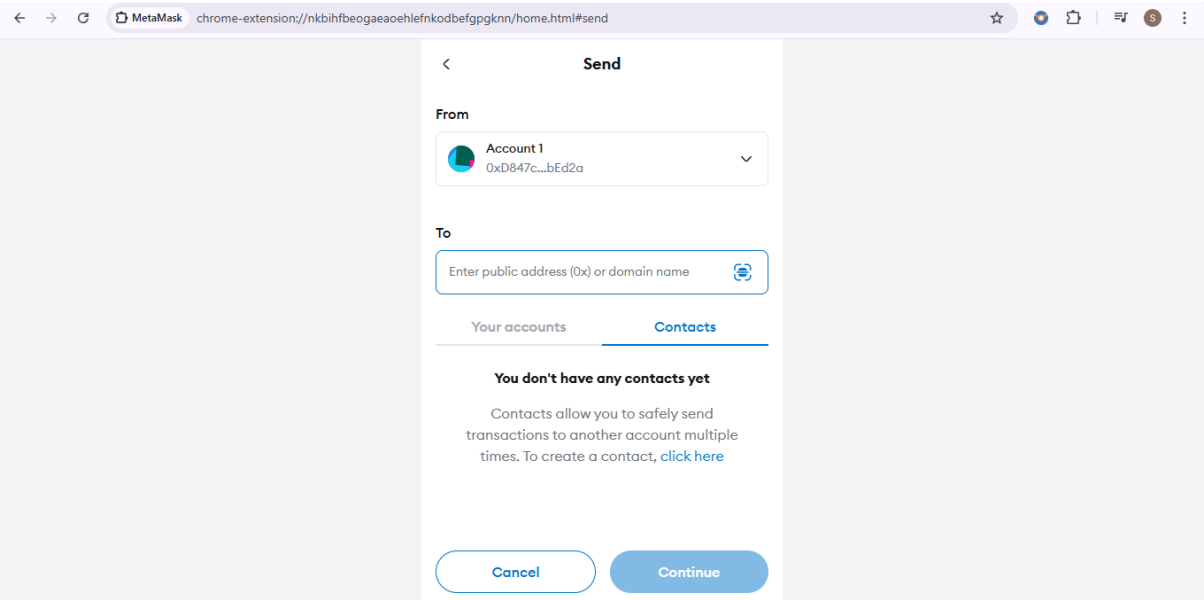
Since Sepolia is a test network, ETH must be obtained from a **faucet**:
1. Visit https://sepoliafaucet.com or https://faucet.quicknode.com.
2. Paste your MetaMask **wallet address**.
3. Click **Request ETH**.
4. Wait for **ETH to be credited** in MetaMask.

Step 5: Send ETH on Sepolia Testnet

1. Open MetaMask and click **Send**.
2. Enter the **receiver's wallet address** (e.g., another test account).
3. Enter the amount (e.g., **0.01 ETH**).
4. Click **Next** → Review transaction details.
5. Click **Confirm** → Wait for confirmation.

Step 6: Receive ETH on Sepolia Testnet

1.Copy your MetaMask **wallet address**.
2. Share it with the sender (or send ETH from another test account).
3. Once sent, check your **MetaMask balance**.



Step 7: Track Transactions on Sepolia Etherscan

1. Visit **Sepolia Etherscan**.
2.Enter your **wallet address** in the search bar.
3.Click on a transaction to view:
Sender & Receiver addresses
Transaction hash
Gas fees & status
Block confirmations

Conclusion:

- MetaMask is a secure **Ethereum wallet** for managing ETH and tokens.
- Sepolia Testnet allows risk-free **testing of transactions**.
- Etherscan helps **track blockchain transactions** for transparency.

Roll No: 160122749301
Experiment No: 04
Sheet No: _____
Date: _____

EXPERIMRNT-04

AIM: Launch Remix web browser and write a smart contract using the solidity language for the ―Hello World program‖.

DESCRIPTION:

n this experiment, we deploy a **Hello World** smart contract written in Solidity on the **Sepolia Testnet** using **Remix IDE** and **MetaMask**. Sepolia is an Ethereum test network that allows developers to test their smart contracts without using real ETH.

This process involves:
1.Setting up **MetaMask** and connecting to the **Sepolia Testnet**.
2.Writing a **Hello World** smart contract in **Solidity**.
3.Compiling the contract in **Remix IDE**.
4.Obtaining **SepoliaETH** from a **faucet** for gas fees.
5.Deploying the contract and verifying it on **Sepolia Etherscan**.
6.nteracting with the deployed contract using **Remix**.

This experiment helps developers understand the **Ethereum development workflow**, including **gas fees, blockchain transactions, and smart contract deployment**.

procedure:

Step 1: Open Remix in a Web Browser

1. Open your web browser (Chrome, Firefox, Edge).
2. Visit **Remix IDE**: https://remix.ethereum.org.
3. Once Remix opens, you will see a file explorer, editor, and terminal.

Step 2: Create a New Solidity File

1. In the **File Explorer** (left panel), click the + button to create a new file.
2. Name the file **HelloWorld.sol**

Step 3: Write the Solidity Smart Contract

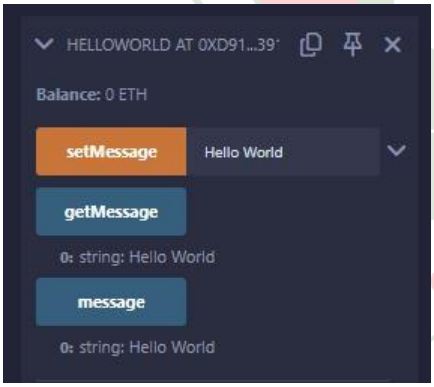
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
    // State variable to store the message
    string public message;

    // Constructor to initialize the message
    constructor(string memory _message) {    infinite gas 352600 gas
        message = _message;
    }

    // Function to change the message
    function setMessage(string memory _message) public {    infinite gas
        message = _message;
    }

    // Function to get the message
    function getMessage() public view returns (string memory) {    infinite gas
        return message;
    }
}
```

Step 4: Compile the Smart Contract

1.In the left panel, click on the **Solidity Compiler** tab (third icon from the top).
2. Select **0.8.0 or a later compiler version** from the dropdown.
3. Click **Compile HelloWorld.sol**.
4. Ensure there are **no compilation errors**.

Step 5: Deploy the Contract to a Test Network

1. Click on the **Deploy & Run Transactions** tab (fourth icon in the left panel).
2. Under **Environment**, select **Injected Provider - MetaMask** (make sure MetaMask is connected to Sepolia Testnet).
3. Click **Deploy**.
4. MetaMask will prompt for confirmation → Click **Confirm**.
5. Once deployed, the contract will appear under **Deployed Contracts** in Remix.



Step 6: Interact with the Smart Contract

1.Click on the deployed **HelloWorld** contract in Remix.
2. Click **getMessage** → It should return `"Hello, World!"`.
3. To change the message:

- Enter a **new message** (e.g., `"Hello, Ethereum!"`) in **setMessage** input.
- Click **setMessage** → Confirm in MetaMask.
- Click **getMessage** again → It should now show the new message.

Step 7: Verify Transaction on Sepolia Etherscan

1. Open **MetaMask**, find the transaction under **Activity**.
2. Click on the transaction to **view it on Sepolia Etherscan**.
3. Check details like **gas fees, sender, contract address, and function calls**.

Conclusion

- We successfully wrote and deployed a **Hello World** smart contract using Solidity on Remix.
- This experiment demonstrated **how to write, compile, deploy, and interact** with a smart contract on Ethereum.
- **MetaMask and Sepolia Testnet** allow risk-free testing before mainnet deployment.
- **Etherscan** is useful for tracking contract execution and transactions.

Roll No: 160122749301
Experiment No: 05
Sheet No: _____
Date: _____

EXPERIMRNT-05

AIM: Write Solidity program

(a) For incrementing/decrementing a counter variable in a smart contract

(b) To send ether from a Meta-mask account to another Meta-mask account through a smart contract.

(c) To simulate a lottery game

DESCRIPTION:

This experiment demonstrates the implementation of a **simple counter smart contract** using **Solidity**. The contract allows users to **increment and decrement** a counter variable on the Ethereum blockchain.

By deploying this contract on a blockchain (e.g., **Sepolia Testnet**), users can interact with the counter while observing how **state changes are stored** and **transactions are executed** on the network.

procedure:

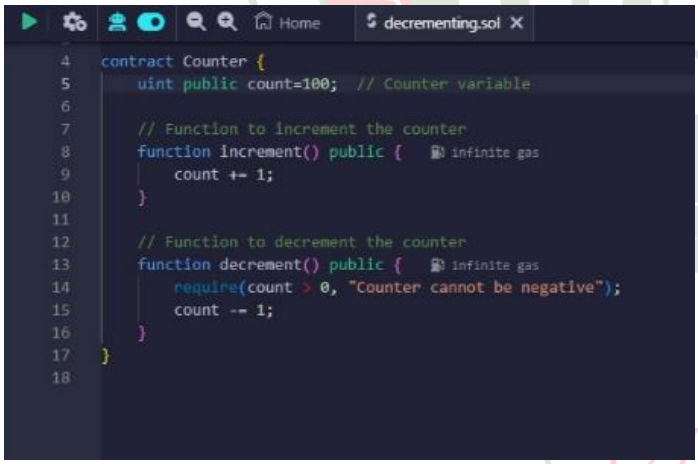
Step 1: Open Remix in a Web Browser

1. Open your web browser (Chrome, Firefox, Edge).
2. Visit **Remix IDE**: https://remix.ethereum.org.
3. Once Remix opens, you will see a file explorer, editor, and terminal.

Step 2: Create a New Solidity File

1. In the **File Explorer** (left panel), click the + button to create a new file.
2. Name the file **Counter.sol**

Step 3: Write the Solidity Smart Contract



Step 4: Compile the Smart Contract

1.In the left panel, click on the **Solidity Compiler** tab (third icon from the top).
2. Select **0.8.0 or a later compiler version** from the dropdown.
3. Click **Compile Counter.sol**.
4. Ensure there are **no compilation errors**.

Step 5: Deploy the Contract to a Test Network

1. Click on the **Deploy & Run Transactions** tab (fourth icon in the left panel).
2. Under **Environment**, select **Injected Provider - MetaMask** (make sure MetaMask is connected to Sepolia Testnet).
3. Click **Deploy**.
4. MetaMask will prompt for confirmation → Click **Confirm**.
5. Once deployed, the contract will appear under **Deployed Contracts** in Remix.

Step 6: Interact with the Smart Contract

1.Compile **the contract using** 0.8.0 or later **compiler version.**
2. Deploy **using** Remix or Sepolia Testnet (with MetaMask)**.**
3. Interact **by calling increment(), decrement(), and getCount().**

Step 7: Verify Transaction on Sepolia Etherscan

1. Open **MetaMask**, find the transaction under **Activity**.
2. Click on the transaction to **view it on Sepolia Etherscan**.
3. Check details like **gas fees, sender, contract address, and function calls**.

Conclusion

In this experiment, we successfully implemented and deployed a **Counter Smart Contract** using Solidity. The contract allows users to **increment, decrement, and retrieve** the counter value, demonstrating **state management in blockchain applications**.

Key Takeaways:

1. Learned **Solidity fundamentals**, including **state variables, functions, and visibility modifiers**.
2. Understood how **transactions modify contract state** and consume **gas fees**.
3. Gained experience in **deploying and interacting with a smart contract** on **Remix IDE**.
4. Observed real-world blockchain features like **immutability and transparency**.

Future Enhancements:

1.**Access Control**: Restrict counter updates to specific users.
2.**Event Logging**: Emit events on every increment/decrement for better tracking.
3.**Smart Contract Verification**: Publish and verify the contract on **Etherscan**.