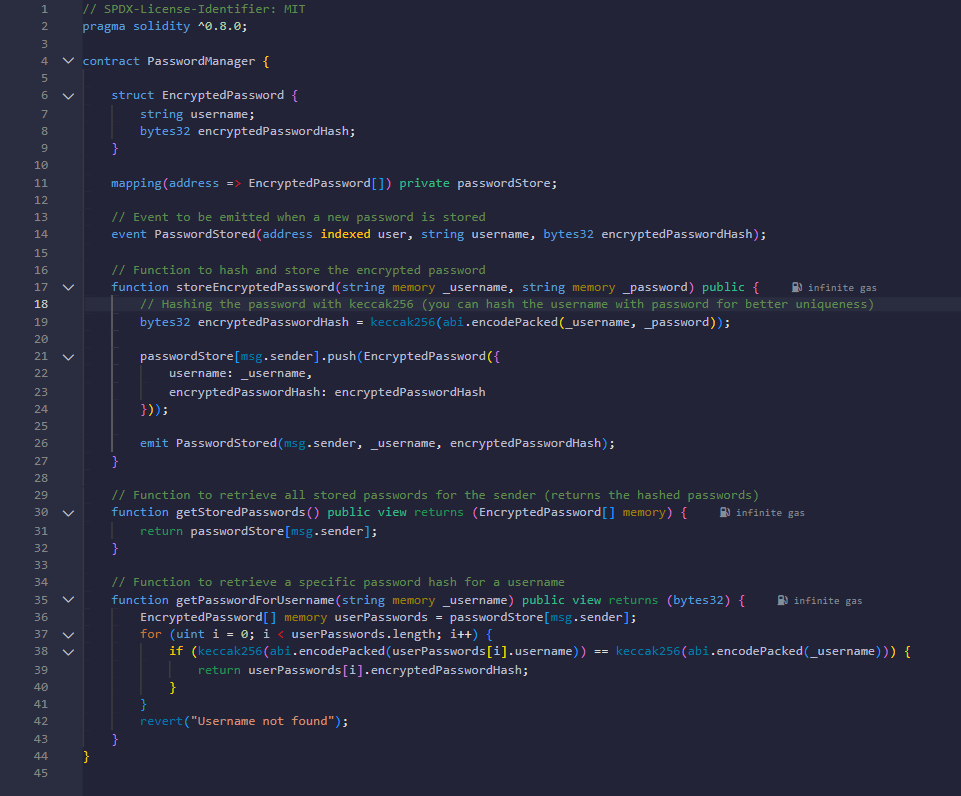
## Work Flow

**Remix - Ethereum IDE:**

After opening Remix ID. In the workspaces create a folder Contracts in that folder add a file **“Password.sol”**. In that file insert the solidity program:

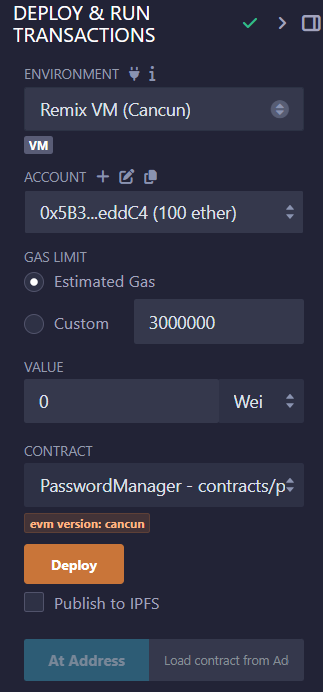
CODE:



The step-by-step process of this Password Manager smart contract workflow:

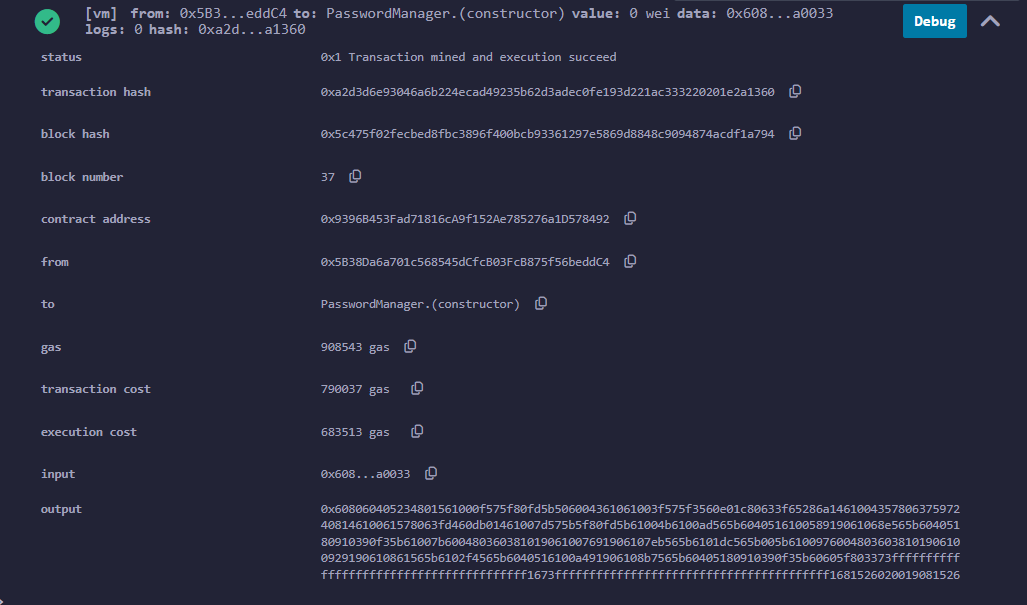
1. Contract Structure:

* The smart contract is named "PasswordManager" using Solidity version ^0.8.8
* It contains a struct called "EncryptedPassword" that stores:
* username (string)
* encryptedPasswordHash (bytes32)



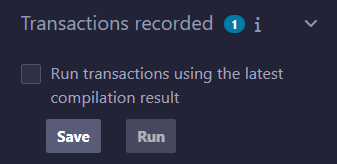
2. Deployment Process:

* Set up the environment in Remix VM (Cancun)
* Account with 100 ether is used for deployment
* Gas limit is set to 3000000
* Deploy the contract using the "Deploy" button



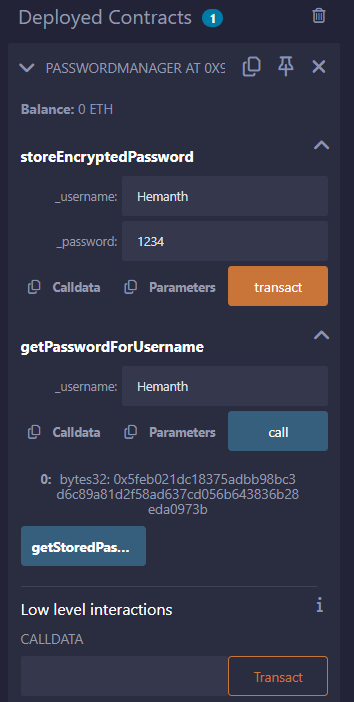
3. Storing a Password:

* Use the `storeEncryptedPassword` function which:
* Takes username and password as inputs
* Hashes the password using keccak256
* Stores it in the passwordStore mapping
* Emits a PasswordStored event



- Example shown in Image 5:

* Username: "Hemanth"
* Password: "1234"
* Click "transact" to store



4. Retrieving Passwords:

The contract provides two retrieval methods:

a) `getStoredPasswords()`

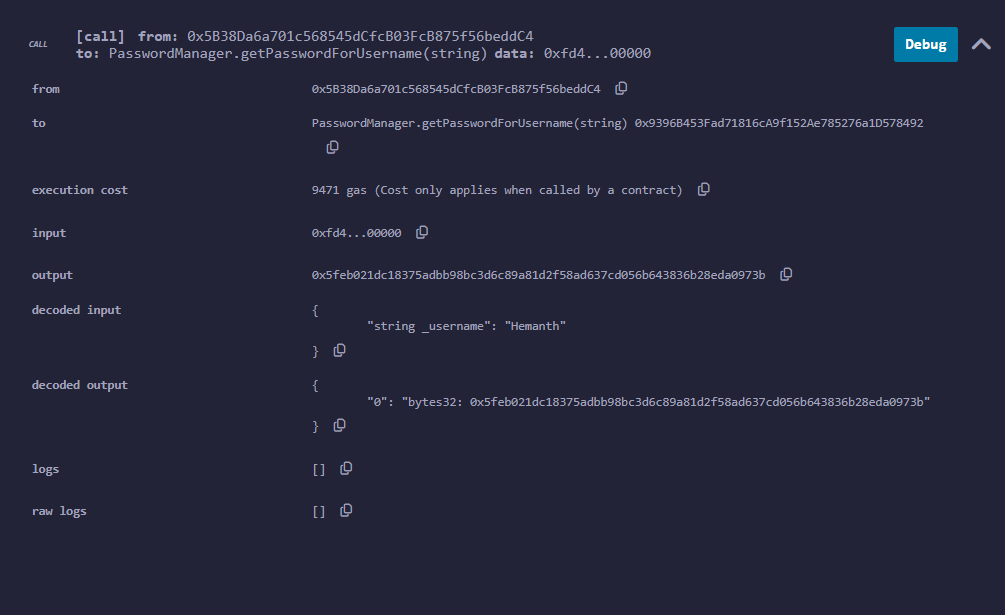
- Returns all stored passwords for the sender

b) `getPasswordForUsername(string memory \_username)`

- Returns the specific password hash for a given username

- Example in Image 5 shows retrieving password for "Hemanth"

- Use the "call" button to execute



5. Transaction Verification:

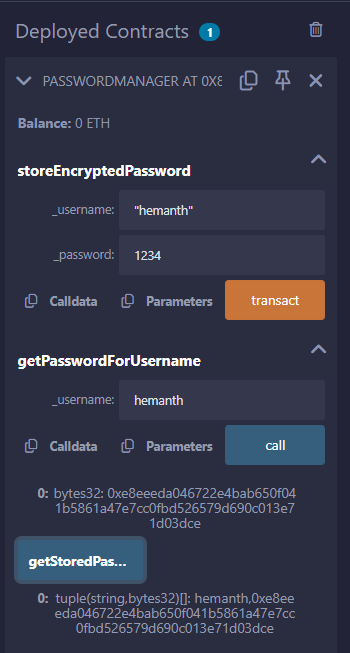
- After deployment/transactions, you can verify:

* Transaction status
* Gas used
* Block details
* Contract address
* Input/Output data

As shown in Image 3

6. Security Features:

* Passwords are encrypted using keccak256 hashing
* Each user's passwords are stored separately
* Private mapping is used for password storage
* Username verification before password retrieval

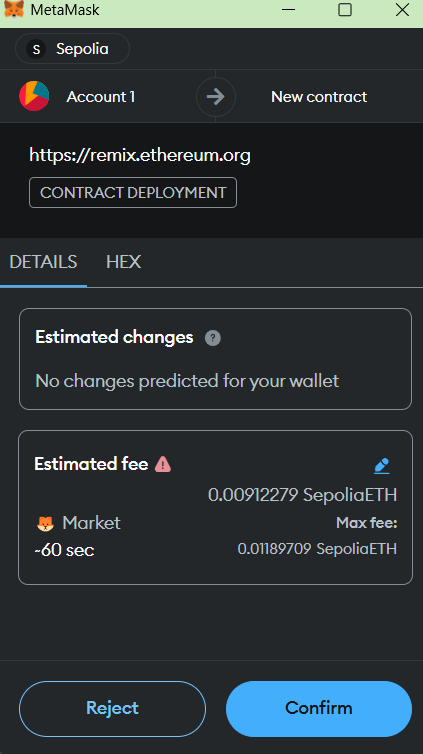


7. User Interface:

- The deployed contract interface shows:

* Function inputs
* Call/Transaction buttons
* Transaction results
* Low-level interaction options

This workflow creates a secure way to store and retrieve encrypted passwords on the blockchain, with proper access controls and encryption in place.



1. MetaMask Connection & Setup:

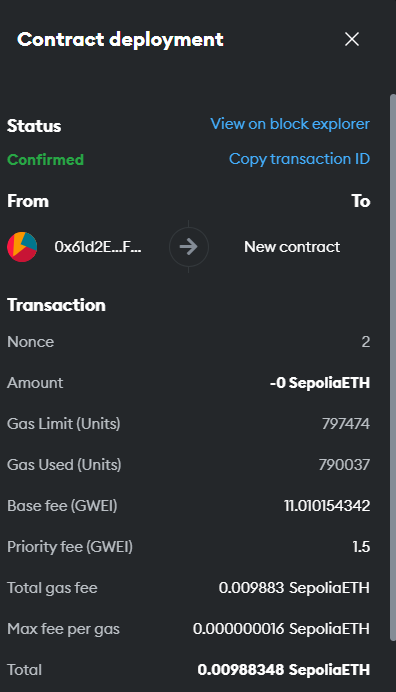
* Open MetaMask wallet
* Connect to Sepolia testnet
* Ensure you have sufficient Sepolia ETH for gas fees
* Account details shown in MetaMask (Account 1)

1. Contract Deployment: a) Initial Setup:
   * Access <https://remix.ethereum.org>
   * Connect MetaMask to Remix
   * Deploy contract with following transaction details:
     + Gas Limit: 797474 units
     + Base fee: 11.01015434 GWEI
     + Priority fee: 1.5 GWEI
     + Total gas fee: 0.009883 SepoliaETH

b) Confirmation:

* Review transaction in MetaMask
* Estimated fee displayed: 0.00912279 SepoliaETH
* Click "Confirm" to deploy

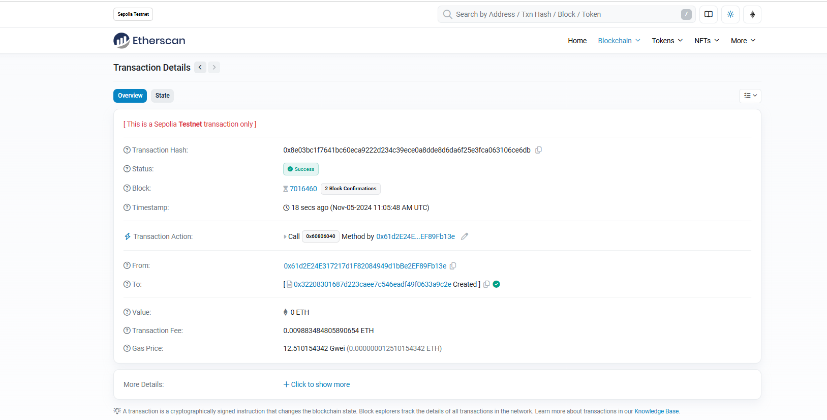
1. Store Password Function: a) Input Data:
   * Open storeEncryptedPassword function
   * Enter username: "hemanth"
   * Enter password: "1234"
   * Click "transact" button



b) Transaction Verification:

* Transaction confirmed on Sepolia network
* View transaction details on block explorer
* Password successfully encrypted and stored

1. Retrieve Password: a) Using getPasswordForUsername:
   * Enter username: "hemanth"
   * Click "call" button
   * Transaction details:
     + Execution cost: 9471 gas
     + Input data shows username: "Hemanth"
     + Output: encrypted password hash returned



1. Transaction Verification on Etherscan:

* Transaction status: Success
* Block: 7016460 (2 Block Confirmations)
* Timestamp: Nov-05-2024 11:05:48 AM UTC
* Transaction fee: 0.00988348480589 ETH
* Gas price: 12.51015434 Gwei
* Contract interaction verified on blockchain

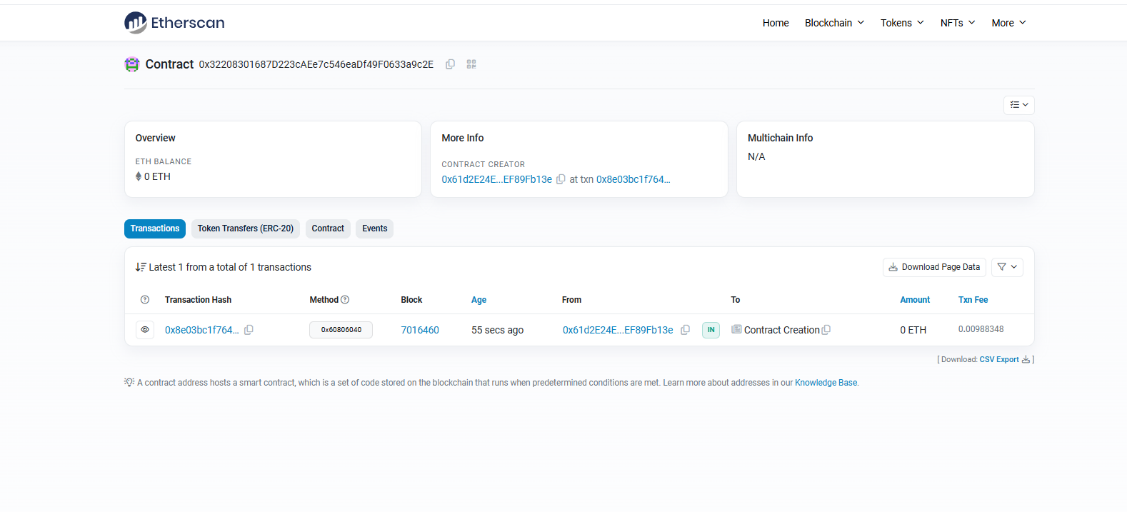
1. Results:

* Contract successfully deployed
* Password stored securely with encryption
* Password retrieval function working correctly
* All transactions confirmed on Sepolia testnet
* Viewable on block explorer for verification

The entire process demonstrates:

* Secure password storage on blockchain
* Successful contract deployment
* Proper function execution
* Transaction verification
* MetaMask integration
* Network interaction on Sepolia testnet

Let me explain the Etherscan contract details shown in the image:



1. Contract Overview:

* Contract Address:0x32208301687D223cAEe7c546eaDf49F0633a9c2E
* ETH Balance: 0 ETH
* Network: Sepolia Testnet (as indicated by earlier images)

1. Contract Information Tabs:

* Transactions
* Token Transfers (ERC-20)
* Contract
* Events

1. Transaction Details:

* Latest Transaction: 1 of 1 total transactions
* Transaction Hash: 0x8e03bc1f764...
* Method ID: 0x60806040 (Contract Creation)
* Block Number: 7016460
* Age: 55 seconds ago at time of snapshot
* From: 0x61d2E24E...EF89Fb13e (Contract Creator)
* To: Contract Creation
* Amount: 0 ETH
* Transaction Fee: 0.00988348 ETH

1. Additional Information:

* Contract Creator: Address starting with 0x61d2E24E...EF89Fb13e
* Created at transaction: 0x8e03bc1f764...
* Multichain Info: N/A

1. Available Actions:

* Download Page Data
* CSV Export option available
* View detailed transaction information
* Toggle between different information views

1. Key Features:

* Real-time tracking of contract transactions
* Transparent view of contract creation details
* Full transaction history available
* Multiple viewing options for contract data
* Verification status and contract interactions visible

This Etherscan page serves as a public ledger showing:

* Contract deployment confirmation
* Transaction verification
* Contract ownership details
* Financial transactions
* Interactive contract status