Summary:

In this experiment, a researcher uses a smart contract to issue a challenge in the blockchain. The contract contains a hashed solution (made using keccak256) and a tiny ETH reward. Any participant can try to solve the challenge by posting an answer. If their answer matches the hash, they will unlock and receive the ETH funds.

Tools used:

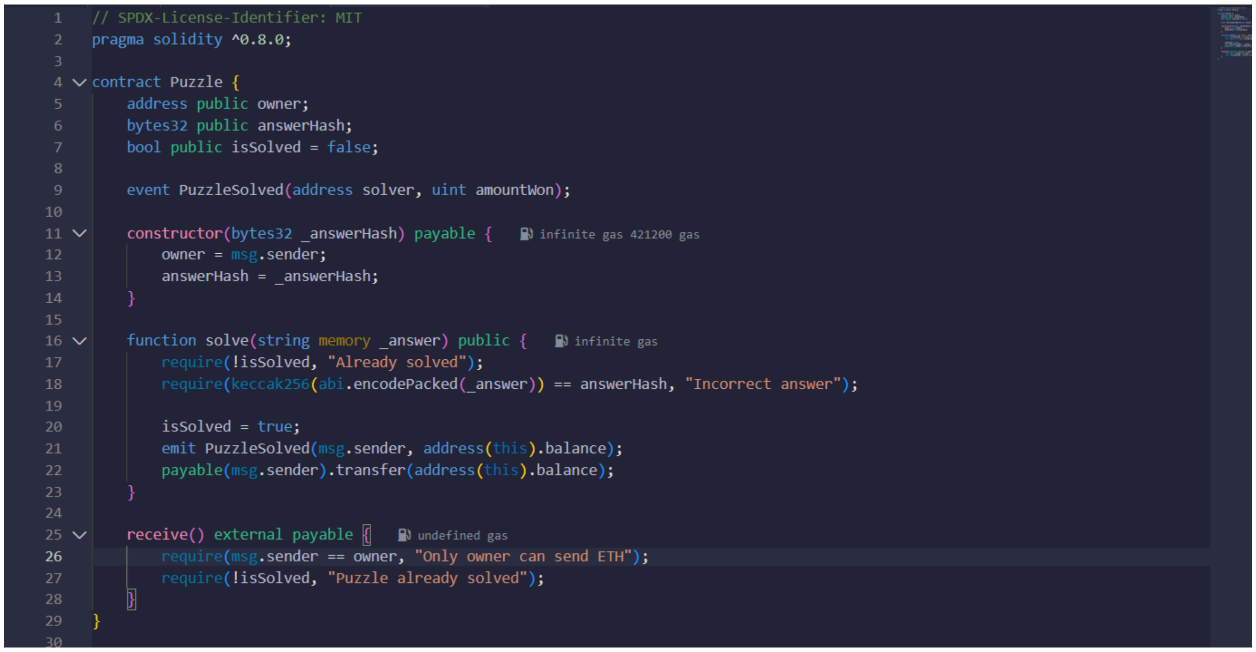
- Remix IDE (VM-Cancun)
- Solidity: 0.8.0
- Web3's keccak256:
    - Used to generate the answering hash:
        - keccak256(abi.encodePacked("42"))

Final Value:
0xccb1f717aa77602faf03a594761a36956b1c4cf44c6b336d1db57da799b331b8

Code Description:

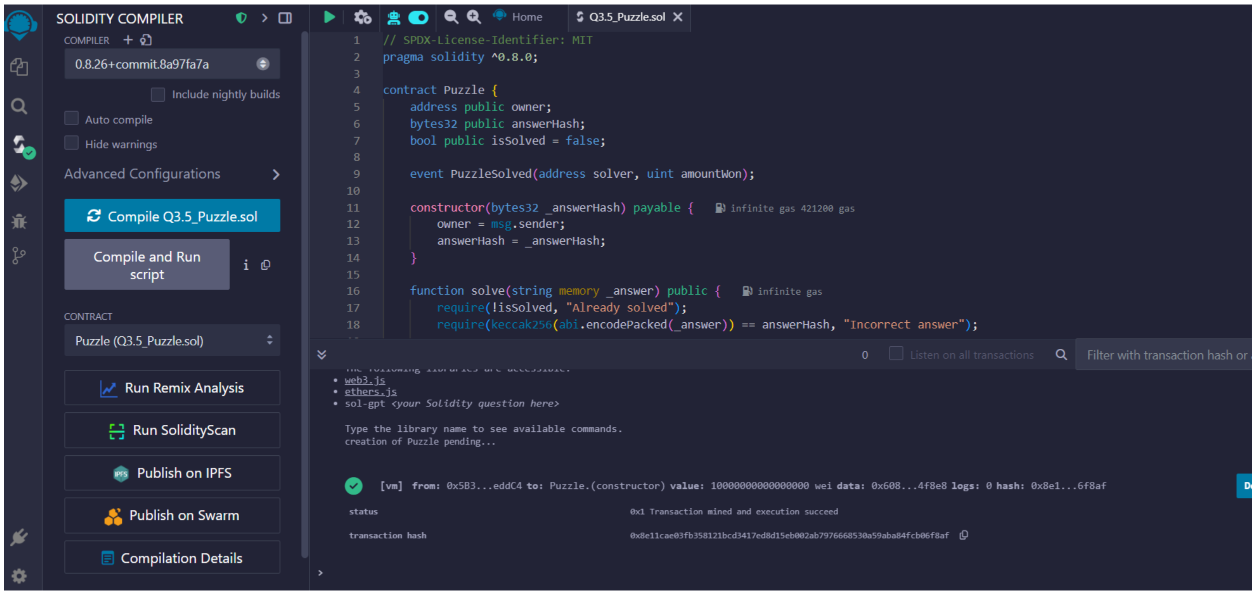
The Smart Contract **Puzzle** is a reward-based problem in which participants must guess a secret answer. The contract is initialized using a keccak256 hash including the expected answer. The **solve()** function lets any guess to be submitted by a user; and if the hashed input matches the stored hash while the puzzle has not previously been solved, the contract delivers its ETH balance to the solver and produces a "**PuzzleSolved**" event. To ensure security and appropriate functionality, the contract prohibits multiple solutions (**isSolved** flag) and limits ETH deposits to the owner solely via the **receive()** function. This ensures one-time solvability and controlled fund management, emulating a fair and verifiable reward system on the blockchain.

Code for Solidity:



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

contract Puzzle {
    address public owner;
    bytes32 public answerHash;
    bool public isSolved = false;

    event PuzzleSolved(address solver, uint amountWon);

    constructor(bytes32 _answerHash) payable {    infinite gas 421200 gas
        owner = msg.sender;
        answerHash = _answerHash;
    }

    function solve(string memory _answer) public {    infinite gas
        require(!isSolved, "Already solved");
        require(keccak256(abi.encodePacked(_answer)) == answerHash, "Incorrect answer");

        isSolved = true;
        emit PuzzleSolved(msg.sender, address(this).balance);
        payable(msg.sender).transfer(address(this).balance);
    }

    receive() external payable {    undefined gas
        require(msg.sender == owner, "Only owner can send ETH");
        require(!isSolved, "Puzzle already solved");
    }
}
```

Solidity Compiler and verification of compiling

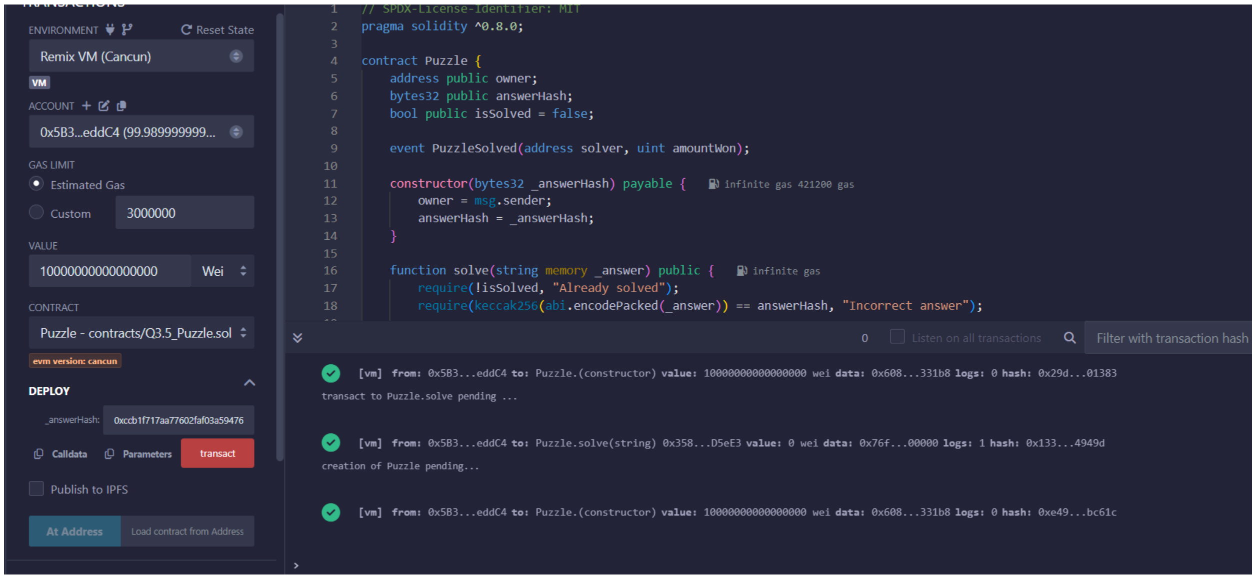


COMPILER: 0.8.26+commit.8a97fa7a
Include nightly builds
Auto compile
Hide warnings
CONTRACT: Puzzle (Q3.5_Puzzle.sol)

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

contract Puzzle {
    address public owner;
    bytes32 public answerHash;
    bool public isSolved = false;

    event PuzzleSolved(address solver, uint amountWon);

    constructor(bytes32 _answerHash) payable {    infinite gas 421200 gas
        owner = msg.sender;
        answerHash = _answerHash;
    }

    function solve(string memory _answer) public {    infinite gas
        require(!isSolved, "Already solved");
        require(keccak256(abi.encodePacked(_answer)) == answerHash, "Incorrect answer");
```

```
 • web3.js
 • ethers.js
 • sol-gpt <your Solidity question here>

 Type the library name to see available commands.
 creation of Puzzle pending...

[vm] from: 0x5B3...eddC4 to: Puzzle.(constructor) value: 10000000000000000 wei data: 0x608...4f8e8 logs: 0 hash: 0x8e1...6f8af
 status                                    0x1 Transaction mined and execution succeed
 transaction hash                          0x8e11cae03fb358121bcd3417ed8d15eb002ab7976668530a59aba84fcb06f8af
```

Deployment & Transaction details and result:

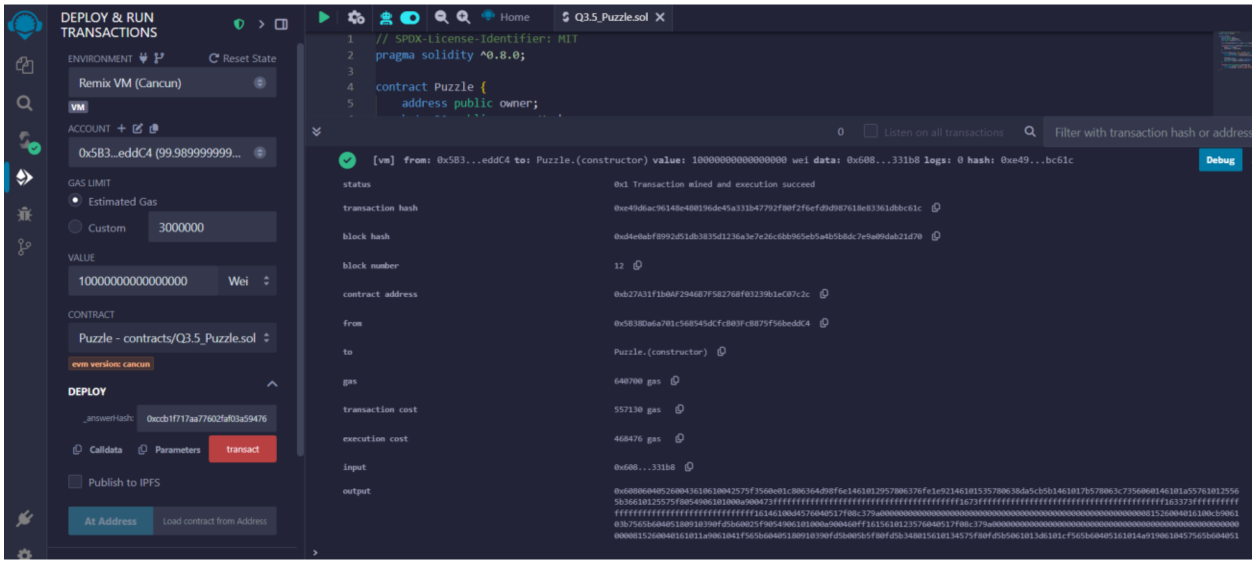

Reset State
DEPLOY
Calldata
Parameters
transact
At Address
Load contract from Address
Listen on all transactions
Filter with transaction hash

ENVIRONMENT: Remix VM (Cancun)
ACCOUNT: 0x5B3...eddC4 (99.989999999...
GAS LIMIT: Estimated Gas; Custom 3000000
VALUE: 10000000000000000 Wei
CONTRACT: Puzzle - contracts/Q3.5_Puzzle.sol
evm version: cancun
_answerHash: 0xccb1f717aa77602faf03a59476
Publish to IPFS

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

contract Puzzle {
    address public owner;
    bytes32 public answerHash;
    bool public isSolved = false;

    event PuzzleSolved(address solver, uint amountWon);

    constructor(bytes32 _answerHash) payable {    infinite gas 421200 gas
        owner = msg.sender;
        answerHash = _answerHash;
    }

    function solve(string memory _answer) public {    infinite gas
        require(!isSolved, "Already solved");
        require(keccak256(abi.encodePacked(_answer)) == answerHash, "Incorrect answer");
```

```
[vm] from: 0x5B3...eddC4 to: Puzzle.(constructor) value: 10000000000000000 wei data: 0x608...331b8 logs: 0 hash: 0x29d...01383
transact to Puzzle.solve pending ...

[vm] from: 0x5B3...eddC4 to: Puzzle.solve(string) 0x358...D5eE3 value: 0 wei data: 0x76f...00000 logs: 1 hash: 0x133...4949d
creation of Puzzle pending...

[vm] from: 0x5B3...eddC4 to: Puzzle.(constructor) value: 10000000000000000 wei data: 0x608...331b8 logs: 0 hash: 0xe49...bc61c
```

Details of verified deployment:

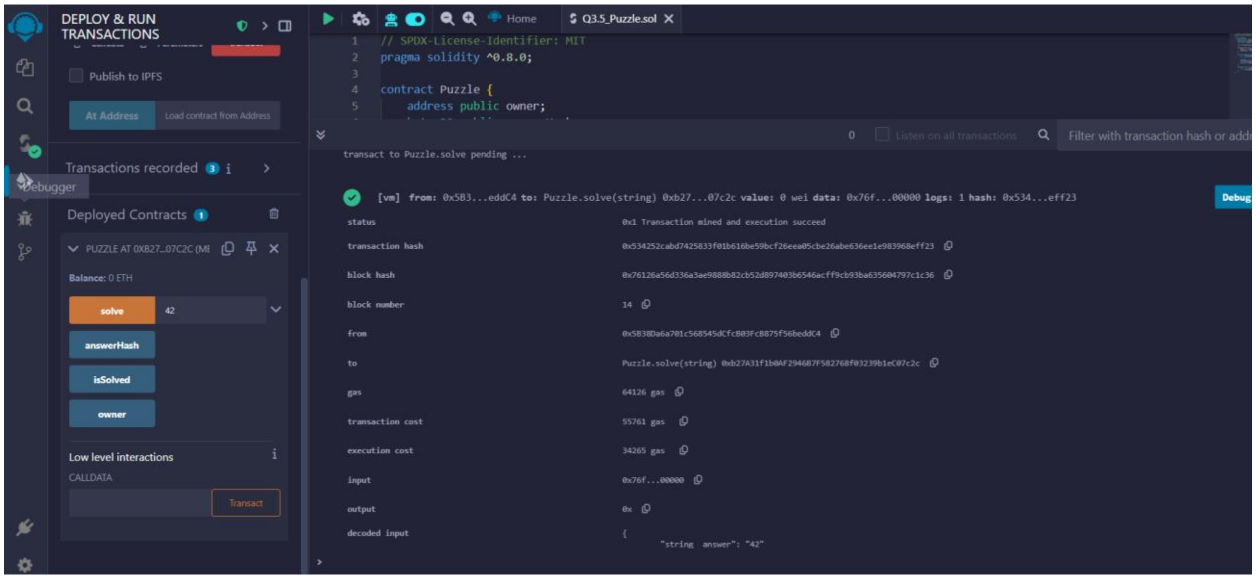

DEPLOY & RUN TRANSACTIONS
Reset State
VM
Calldata
Parameters
transact
At Address
Load contract from Address
Home
Q3.5_Puzzle.sol
Listen on all transactions
Filter with transaction hash or address
Debug

ENVIRONMENT: Remix VM (Cancun)
ACCOUNT: 0x5B3...eddC4 (99.989999999...
GAS LIMIT: [✓ Estimated Gas] Custom 3000000
VALUE: 100000000000000000 Wei
CONTRACT: Puzzle - contracts/Q3.5_Puzzle.sol
evm version: cancun
DEPLOY
_answerHash: 0xccb1f717aa77602faf03a59476
Publish to IPFS

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

contract Puzzle {
    address public owner;
```

```
[vm] from: 0x5B3...eddC4 to: Puzzle.(constructor) value: 100000000000000000 wei data: 0x608...331b8 logs: 0 hash: 0xe49...bc61c
status                 0x1 Transaction mined and execution succeed
transaction hash       0xe49d6ac96148e480196de45a331b47792f80f2f6efd9d987618e83361dbbc61c
block hash             0xd4e0abf8992d51db3835d1236a3e7e26c6bb965eb5a4b5b8dc7e9a09dab21d70
block number           12
contract address       0xb27A31f1b0AF294687F582768f03239b1eC07c2c
from                   0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                     Puzzle.(constructor)
gas                    640700 gas
transaction cost       557130 gas
execution cost         468476 gas
input                  0x608...331b8
output                 0x608060405260043610610042575f3560e01c806364d98f6e1461012957806376fe1e9214610153578063 8da5cb5b1461017b578063c7356060146101a55761012556
                       5b36610125575f8054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffff
                       ffffffffffffffffffffffffffffff16146100d4576040517f08c379a000000000000000000000000000000000000000000000000000000000815260040161 00cb9061
                       03b7565b60405180910390fd5b60025f9054906101000a900460ff1615610123576040517f08c379a00000000000000000000000000000000000000000000000000000
                       0000815260040161011a9061041f565b60405180910390fd5b005b5f80fd5b348015610134575f80fd5b5061013d6101cf565b60405161014a9190610457565b604051
```

Solver Test & Result, correct hash:

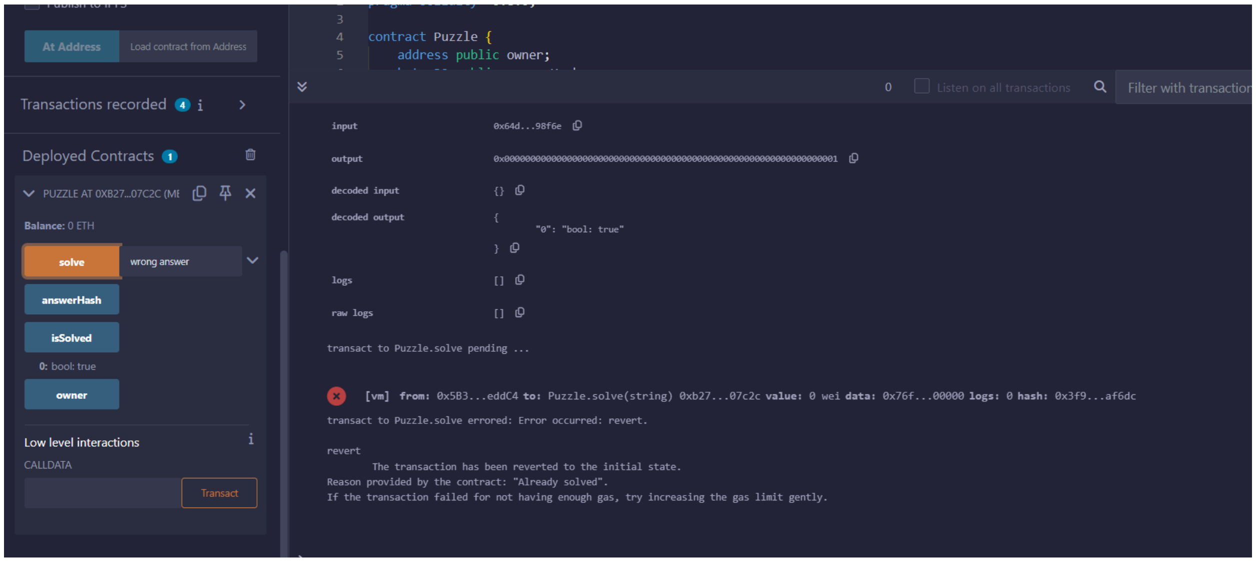


Balance: 0 ETH
solve: 42

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

contract Puzzle {
    address public owner;
```

```
transact to Puzzle.solve pending ...

[vm] from: 0x5B3...eddC4 to: Puzzle.solve(string) 0xb27...07c2c value: 0 wei data: 0x76f...00000 logs: 1 hash: 0x534...eff23
status                 0x1 Transaction mined and execution succeed
transaction hash       0x534252cabd7425833f01b616be59bcf26eea05cbe26abe636ee1e983968eff23
block hash             0x76126a56d336a3ae9888b82cb52d897403b6546acff9cb93ba635604797c1c36
block number           14
from                   0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                     Puzzle.solve(string) 0xb27A31f1b0AF294687F582768f03239b1eC07c2c
gas                    64126 gas
transaction cost       55761 gas
execution cost         34265 gas
input                  0x76f...00000
output                 0x
decoded input          {
                              "string answer": "42"
```

Solver Test & Result: Wrong Answer



```
contract Puzzle {
    address public owner;
```

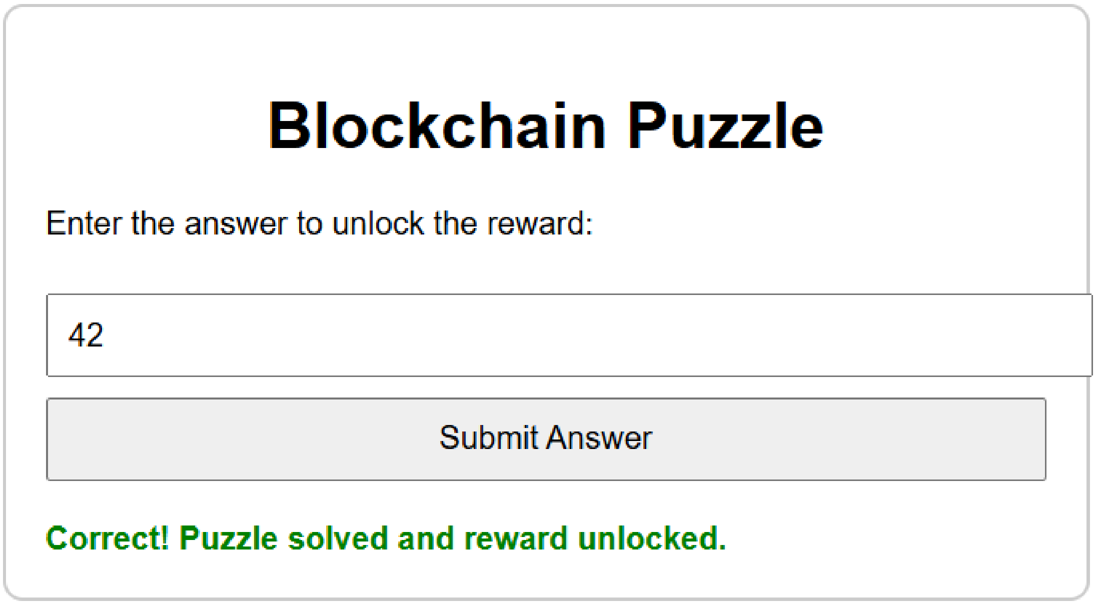
```
input            0x64d...98f6e
output           0x0000000000000000000000000000000000000000000000000000000000000001
decoded input    {}
decoded output   {
                      "0": "bool: true"
                 }
logs             []
raw logs         []

transact to Puzzle.solve pending ...

[vm] from: 0x5B3...eddC4 to: Puzzle.solve(string) 0xb27...07c2c value: 0 wei data: 0x76f...00000 logs: 0 hash: 0x3f9...af6dc
transact to Puzzle.solve errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Already solved".
If the transaction failed for not having enough gas, try increasing the gas limit gently.
```

Html Code for Front end Website:

Front end:

```
<!DOCTYPE html>

<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
 <title>Puzzle Frontend</title>
 <script src="https://cdn.jsdelivr.net/npm/js-sha3@0.8.0/src/sha3.min.js"></script>
 <style>
  body {
   font-family: Arial, sans-serif;
   max-width: 500px;
   margin: 40px auto;
   padding: 20px;
   border: 2px solid #ccc;
   border-radius: 10px;
  }
  h1 {
   text-align: center;
  }
  input, button {
   padding: 10px;
   font-size: 1rem;
   margin-top: 10px;
   width: 100%;
  }
  #status {
   margin-top: 20px;
   font-weight: bold;
  }
 </style>
</head>
<body>
 <h1>Blockchain Puzzle</h1>
 <p>Enter the answer to unlock the reward:</p>

 <input type="text" id="answerInput" placeholder="Type your answer..." />
 <button onclick="submitAnswer()">Submit Answer</button>

 <div id="status"></div>

 <script>
  const correctHash =
"0xccb1f717aa77602faf03a594761a36956b1c4cf44c6b336d1db57da799b331b8";

  function submitAnswer() {
   const answer = document.getElementById("answerInput").value;
   const hashHex = "0x" + keccak256(answer);

   const statusEl = document.getElementById("status");
   if (hashHex === correctHash) {
    statusEl.innerHTML = "Correct! Puzzle solved and reward unlocked.";
    statusEl.style.color = "green";
   } else {
    statusEl.innerHTML = "Incorrect answer. Try again.";
    statusEl.style.color = "red";
   }
  }
 </script>
</body>
</html>
```

Positive answer on the front end:



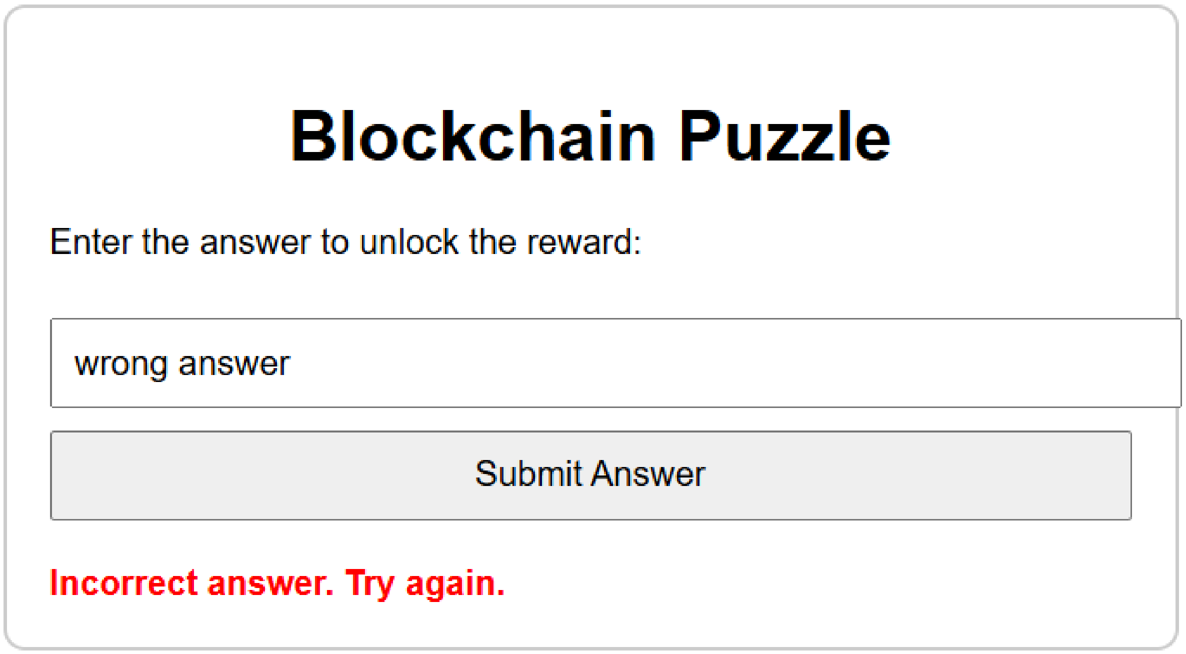
Blockchain Puzzle

Enter the answer to unlock the reward:

42

Correct! Puzzle solved and reward unlocked.

Negative attempt on the Puzzle:



Blockchain Puzzle

Enter the answer to unlock the reward:

wrong answer

Incorrect answer. Try again.

References

Ethereum Foundation. (n.d.). *Remix IDE: Web-based tool for Solidity development*. [online] Available at: https://remix.ethereum.org [Accessed 11 Apr. 2025].

Solidity Team. (2025). *Solidity 0.8.x Documentation*. Ethereum Foundation. [online] Available at: https://docs.soliditylang.org/en/latest/ [Accessed 13 Apr. 2025].

Chen, Z. (2020). *js-sha3: A simple JavaScript implementation of SHA3 (Keccak)*. [online] GitHub. Available at: https://github.com/emn178/js-sha3 [Accessed 16 Apr. 2025].

Mozilla Developer Network. (n.d.). *HTML: HyperText Markup Language*. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/HTML [Accessed 17 Apr. 2025].

Mozilla Developer Network. (n.d.). *CSS: Cascading Style Sheets*. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/CSS [Accessed 18 Apr. 2025].

Mozilla Developer Network. (n.d.). *JavaScript Guide*. [online] MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide [Accessed 22 Apr. 2025].