Smart Contracts: Coin Guess:

Discussion Question:

Potential Solution:

Collecting collateral in advance is a step in the right direction but doesn't solve the trust problem, still need an enforcer of some sort.

Trusted centralised entities (governments, courts, companies with good track record, a trusted friend etc) are the best solution for problems like these

Both players physically:

- It's difficult to automate, could have a robot flip the coin but that might be too expensive and time consuming to set up and we still need to trust the robot's code

-Having a virtual coin flip by writing a computer program for example still requires us to trust the code (even with a code audit, we need to trust that the compiler and the hardware)

Online: even worst, at least physically you can verify the coin flip – online even this can be faked (clever video editing, deep fakes etc). In the roulette table of an online casino for example, how can you be sure that the live feed video you see is actually live and not many pre-recorded videos that are changed after your bet to make you lose?

Discussion Question:

Potential Solution: We haven't really formally defined trust, and saying that blockchain systems eliminate trust is not entirely true, depending on the definition of trust it can be argued that blockchains just provide a shift in the nature of trust. We still need to trust some components of a blockchain system (e.g. network of miners)!

"Rather than assuming it to abolish (interpersonal) trust, this line of studies rather argues for a *shift* of the nature of trust by blockchain, replacing interpersonal trust with trust (or: confidence, see De Filippi et al., 2020) in the distributed ledger itself (miners, consensus mechanisms, nodes), software developers (Walch, 2019) or new intermediaries (e.g. crypto-currency exchanges in Brekke, 2019, pp. 83-84)." (https://policyreview.info/glossary/trust-blockchain)

Question 1:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

// contract for coinfip (or guess) that uses the first contract

contract CoinFlip {

}
```

Question 2:

```
// get the hash only using solidity - Warning: don't run this on-chain as your choice will be visible!

// this is using the "pure" modifier as it doesn't read or write to storage

function getHash(bool choice, uint256 nonce) external pure returns (bytes32) {

      return keccak256(abi.encode(choice, nonce));

}
```

Question 3:

```
    // player address variables - set player addressees to zero

    address payable player1 = payable(0x0);

    address payable player2 = payable(0x0);

    // player selection variables

    bytes32 p1selection;

    bool p2selection;

    // bet and expiration variables

    uint256 public bet;

    uint256 public expiration = 2**256-1; // set to max
```

Question 4:

```
// player1 sets choice by giving a hash (hash = H(choice + nonce))

    function makeBet(bytes32 hash) external payable {

        // check if the player has already played (i.e. address not zero)

        require(player1 == payable(0x0));

        // check if bet > 0

        require(msg.value > 0);

        // get address of player 1

        player1 = payable(msg.sender);

        // save player1 selection

        p1selection = hash;

        // store the bet amount
```

```
        bet = msg.value;
    }
```

Question 5:

```
    // player2 sets choice
    function takeBet(bool choice) external payable {
        // check if the player has already played (i.e. address not zero)
        require(player2 == payable(0x0));
        // check if player 1 made their choise
        require(player1 != payable(0x0));
        // check if the bet is same as player 1
        require(msg.value == bet);
        // save address of player 2
        player2 = payable(msg.sender);
        // save player 2 choice
        p2selection = choice;
        // set expiration date
        expiration = block.timestamp + 24 hours;
    }
```

Question 6:

```
    // find out who won, player1 provides choice(0/1) and nonce(32 byte unsigned
int) that match stored hash
    // note that the address of player doesn't need to be checked here, as only
player should know the correct nonce
    function reveal(bool choice, uint256 nonce) external {
        // check if H(choice + nonce) is equal to stored hash
        require(keccak256(abi.encode(choice, nonce)) == p1selection);
        // check if second player played
        require(player2 != payable(0x0));
        // players made the same choice => 2 wins
        if (p2selection == choice) {
```

```
            // player2 gets ether
            player2.transfer(address(this).balance);
        }
        // players made different choices => 1 wins
        else {
            // player1 gets ether
            player1.transfer(address(this).balance);
        }
    }
```

Question 7:

```
    // player 1 can cancel bet if player2 hasn't played yet
    function cancel() external {
        // check if it is player1
        require(msg.sender == player1);
        // check if player2 hasn't played yet
        require(player2 == payable(0x0));
        // player1 gets bet back
        player1.transfer(address(this).balance);
    }
```

Question 8:

```
      // if player1 doesn't reveal their choice before the bet expires, palyer2
can timeout and win
    function timeout() external {
        // check if it has expired
        require(block.timestamp >= expiration);
        // player2 gets both bets
        player2.transfer(address(this).balance);
    }
```