467H - Principles of Decentralized Ledgers
Smart Contracts Coursework

Abraão Pacheco Dos Santos Peres Mota - **CID:** 00941232

1 Introduction

I have opted to write a smart contract implementing a game of rock paper scissors. Due to their existence in the blockchain, as well as no enforced guarantees as to when a player must take their turn, this otherwise simple game becomes more complex due to security concerns.

2 Game Architecture

The game has 4 distinct stages, each of which are separated by user driven actions. These comprise:

1. **Game registration** - This is the first step taken in the game. Before the game proceeds, both players must have registered their interest in partaking in the game. This stage allocates the callers `address` to be set to either Player 1 or Player 2, if either of these are available at the moment. If they are not available, the contract caller is not considered a valid player, so any attempts at calling any subsequent steps will fail. Registration comes at a price - the players must provide a minimum entry price to enter the game (like a bet), so that the winner may receive a price and monetary rewards can be duly distributed to the miners of the transactions involved in the game. This contract only allows for 1 game at any given time, so any existing game must finish before any other game can commence.
2. **Move commitment** - Due to security concerns of using the blockchain, the game follows a *Commit-Reveal* approach. In this stage of the game, the player commits to a specific move. They do this by submitting a signature to the game, comprised of the `keccak256` hash of the string made up by `move + salt`, where the `move` is one of `"rock"`, `"paper"` or `"scissors"`. The `salt` is a random string of the players choice used to hash their move. This salt should be kept secret until the reveal stage of the game.
3. **Move reveal** - At this stage, the player reveals their move of choice to the contract. They do so by submitting their `move` and `salt` in plaintext when calling the `revealMove` function. After this submission, the contract checks whether this is the valid move that the player has committed to previously; if the signature that was submitted in the commit phase does not match the `keccak256` hash of the `move` and the `hash` that were passed in at this stage, then the player is seen to be attempting to cheat - they have committed to a signed move but when revealing they attempt to provide a different move.
4. **Checking for end of game** - This function checks for the sufficient conditions to finish the game in question. This can happen in 1 of 2 ways - either both players have revealed their moves within a reasonable threshold of one another, or one player has waited a sufficiently long time after revealing (in this contract, the waiting threshold is set to 5 minutes). If one of these conditions has been met, then the game is considered to be over, and the winner is decided. If any player has been considered to be cheating at any point in the game, they will not receive any of the winnings (unless both players have cheated, in which case the game is considered a draw, and the winnings are split equally between the two).

3 Threat Modelling

As it stands, the game must assume the worst of both players to be able to execute in a fair way that rewards players appropriately. There are a number of different ways in which a player could attempt to cheat the system.

- **Read opponents move or change your own move** - This is the biggest security concern of implementing this game in the blockchain. All the information on the blockchain is stored in full sight of every user, and the smart contract writer cannot guarantee that both players play their move at the same time - as soon as a user submits their move, the transaction containing this information becomes public. The opponent could inspect this transaction and thus change their move accordingly to win the game everytime. Due to this, a *Commit-Reveal* approach must be used to safeguard both parties. This works by ensuring that both players commit to a move before they reveal their move to the blockchain. The commit phase is done by submitting a signature that fits a pre-defined format (in our case, it is the output of running `keccak256(move + salt)` in *Solidity*). Because the *salt* is a player chosen attribute, the user can submit the signature without letting others know what his move is. The *salt* can then be used in the reveal phase to verify the signature. The *Commit-Reveal* approach relies on the fact that reversing the produced `keccak256` signature (finding the inputs required to generate the signature given) is computationally infeasible for the game players. If this could be reversed, then the move could be obtained as soon as the signature is uploaded to the blockchain, rendering this approach useless in securing the game against cheaters.

- **Denial of Service** - Given we are using the *Commit-Reveal* approach, a malicious player could still attempt to ruin the game. Suppose that both players have committed to a a move, and then Player 1 reveals his move. This has made his move public to Player 2, and may reveal to Player 2 that he will lose the game (e.g. Player 1 has picked *"rock"* and Player 2 is *"scissors"*). Player 2 may then choose to deny the end of the game from happening by choosing to never reveal his move. In order to decentivise this deviant behaviour, the game sets a maximum waiting threshold of 5 minutes after the first move reveal by any player for the other player to also reveal their move. If the second revealer still hasn't revealed his move after this time threshold has elapsed when the `checkGameEnd` function is called, then this player is declared to be a cheater.

- **Tamper with existing games** - Given that the contract is in the public domain,

If for example Player 1 has revealed his move, and 5 minutes after this event Player 2 still hasn't revealed their move, then it is assumed that Player 2 is cheating because they may have read the move of the other player and are attempting to

Honesty analysis

4 Extensions and Further Concerns

- Web interface (make sure keccakimplementation is the same) - Minimum length salt length - Paying contract maker - Multiple games at once