P2P Project Report

Emanuele Buonaccorsi - 598855

Overview

The main components of this project are:

- A **smart contract** (`Mastermind.sol`)
- A **web app** to interact with the contract (contained in the `/frontend/` folder)
- A **script** to **deploy** (`/scripts/deploy.js`)
- A **script** to **transfer** currency to a certain account (`/tasks/faucet.js`)
- A **test** suite

You can find the public repository for the project on my GitHub profile.

The Hardhat starter project was used as a base to get started with the project structure. For the client, I implemented a web app using React to build the user interface and JavaScript to implement the logic. To store the private key, I've used a wallet and integrated with it. I've tested the client using Chrome with the Metamask extension, but other wallets should work too.

User Manual

This section guides the user to set up and run the project.

Note: For this project, I've used `pnpm` as a package manager, but `npm` should work fine too by just replacing "pnpm" with "npm" in the commands.

Deploying the Contract

First thing: **install the dependencies**

```
# While in the root of the project
pnpm install
```

Once installed, let's **run Hardhat's testing network**:

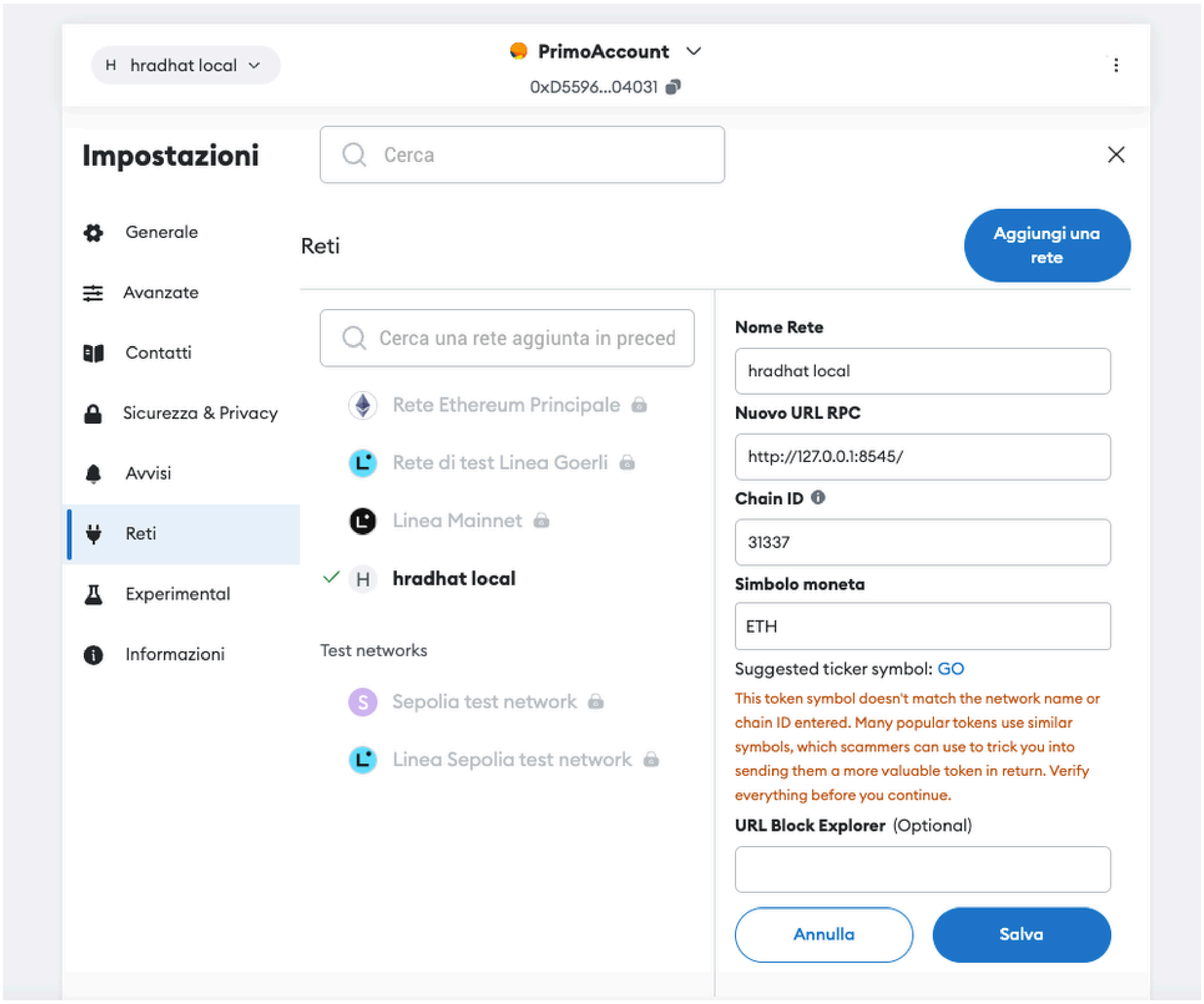
```
npx hardhat node
```

Deploy the contract: On a new terminal, go to the repository's root folder and run this:

```
npx hardhat run scripts/deploy.js --network localhost
```

Installing the Wallet on the Browser

- Make sure to have Coinbase Wallet or Metamask installed on your browser (and listening to `localhost 8545`).
- You'll need to add the Hardhat local test network to Metamask's networks via Metamask's settings (you can reach them by opening the browser extension pop-up).



Nome Rete: hradhat local

Nuovo URL RPC: http://127.0.0.1:8545/

Chain ID: 31337

Simbolo moneta: ETH

Suggested ticker symbol: GO

This token symbol doesn't match the network name or chain ID entered. Many popular tokens use similar symbols, which scammers can use to trick you into sending them a more valuable token in return. Verify everything before you continue.

URL Block Explorer (Optional)

Web App

After installing and configuring a wallet, we can finally run the web app by entering the `frontend` folder and running:

```
cd frontend
pnpm install # Install the dependencies for the frontend: they are different from the r
pnpm run start # Serve the web app on localhost
```

- Now, open http://localhost:3000/ to see your React DApp run on the browser.
- You can now create or join a private or public game.
- If needed, you can see more messages by opening the console in the developer tools (F12 on Chrome).

Give Money Using the Faucet Script

- You'll probably need money on your account.
- To do so, you can use the faucet script that can be found in `/tasks/faucet.js` , by running:

```
npx hardhat --network localhost faucet <address>
```

For convenience, you can also copy a ready-to-go command from the web app.

Implementation Details

Here we discuss the main decisions made to implement this project.

Web App

- I've built a web app using the frontend library React.
- I've used `ethers.js` to interact with the smart contract.
- The main component used as an entry point can be found in the file `/frontend/src/components/Dapp.js` .
- For simplicity during debugging, the web app listens for all the events, even if inefficient. A better alternative would be to use the built-in event filters.
- The web app has many game states. Some events trigger a transition from one event to another.
- At the end of any turn, the client shows the option to select one of the feedbacks for a dispute.
- When working with secret codes in the frontend, you'll work with a series of color circles. During the communication with the smart contract, the colors are converted to a text code in the form `RGBY` .

Contract

Games

- A game is represented as a struct.
- Every game has a `GameState` : Created, Joined, InProgress, Ended.
- When the game is `InProgress` , even the turn has a state that helps determine what we expect the players to do next: `Commit` , `Guess` , `Feedback` , `Reveal` , `WaitingForDispute` .

Game Creation

- The user that wants to create a game has to call `createGame()` .

- createGame can create a **private or public game**:
    - **A private game** can only be joined by the user specified by the creator.
    - **A public game** can be joined by anyone, and is inserted into the `gamesWithOnePlayer` array, where it can be extracted randomly.

Joining a Random Game

To implement the extraction for a random game in an efficient way, I've used two data structures to keep track of the current games available:

```
mapping(uint => uint) private gameIdToIndex;
uint[] private gamesWithOnePlayer;
```

I then extract one of these games pseudo-randomly:

```
uint randomIndex = uint(keccak256(abi.encodePacked(block.timestamp, block.prevrandao, s
```

Reveal and Dispute Implementation

- To avoid cheating, when the codemaker reveals their code, the smart contract checks that the revealed code matches the hash that they previously committed. If there is no correspondence, the reveal function fails.
- The smart contract keeps track of the guesses and the feedbacks given for the current turn.
- At the end of a turn, after the codemaker revealed their code, the codebreaker has to call one of these 2 functions: `dispute` or `dontDispute` .
    - If they choose `Dispute` :
        - They have to pass the number of the guess they want to dispute.
        - The smart contract then computes the real feedback.
        - If the feedback was correct, the codemaker wins; otherwise, the codebreaker wins.
    - If the codemaker chooses `dontDispute` :
        - The points are assigned.
        - The game moves on to the next turn, if left.

AFK Implementation

- A player can accuse of AFK only when waiting for the other player.
- **To accuse, the player has to call `startAccuseAFK` :** When calling this function, the smart contract saves the current timestamp (the block number, actually) as an `accusationTimestamp` .
- The accused user receives a notification, to remember them to make a move.

- **To assign the penalty, the same player has to call `endAccuseAFK` after a while.** There are 3 cases:
    - **The other user still has time:** If the other player has some time left to make a move, you cannot `endAccuseAFK`. You can call it only after `TIME_DISPUTE_BLOCKS` after the accusation, so the accuser should try again after a while.
    - **The other user made a move in time:** `accusationTimestamp` is reset, and the accuser will not be able to call `endAccuseAFK` and give them a penalty.
    - **The other user is AFK and didn't make a move in time:** the game ends, and the accuser wins.

`TIME_DISPUTE_BLOCKS` has been intentionally chosen with a small value to make testing faster.

Potential Vulnerabilities

In this section, we analyze potential vulnerabilities and/or the measures used to mitigate them.

Randomness

- In some parts of the project, pseudo-randomness is used.
- For example, to extract a random game, I've used the following code:

```
uint randomIndex = uint(keccak256(abi.encodePacked(block.timestamp, block.prevrandao, s
return gamesWithOnePlayer[randomIndex];
```

Since smart contracts are deterministic, and these parameters can be manipulated by the validators, this code doesn't guarantee true randomness.

A better option for randomness could be using an oracle off-chain.

For simplicity, since the randomness used in this project is not crucial (i.e. it is not important who starts as a codemaker during the first turn), I've decided to stick with pseudo-randomness.

Currency Transfers

To avoid re-entrancy attacks, `address.call.value()` is never used in the contract, preferring `send()` or `transfer()`, as they employ a gas limit that helps mitigate this vulnerability.

Overflow and Underflow

- Integer overflow and underflow are currently possible in some of the arithmetic operations used in the contract.

To mitigate these vulnerabilities, it would be recommended to use the `SafeMath` library and/or assertions.

Authentication Using `msg.sender`

- Modifiers like `onlyPlayers` and `require()` statements are employed to avoid some functions being called by unauthorized users.
- `tx.origin` is never used for authentication to prevent phishing attacks.
- `msg.sender` is used instead.

Tests and gas report

Here follows the output of the tests and the gas report.

Note: due to hardcoding of the codeMaker address, if running the tests, it might be necessary to run it a couple of times untill they all pass. This is because the codeMaker is randomly extracted by the contract.

```
Hardhat configuration loaded!

  Mastermind contract
    Deployment
      ✔ Should deploy with correct initial values
    Game Management
      ✔ Should create a publicgame and emit GameCreated event
      ✔ Should join a private game and emit GameJoined event
      ✔ Should start a game and emit GameStarted event
      ✔ Should return 'This game is reserved for another opponent' error
      ✔ Should make a guess and emit CodeGuessed event
      ✔ Should give feedback and emit CodeGuessedSuccessfully or CodeGuessedUnsuccessfu
      ✔ Should reveal code, emit CodeRevealed event and be left with 1 turn and 3 guess
      ✔ Should handle disputes correctly and emit DisputeVerdict event

·------------------------------------|--------------------------|-------------|--------
|       Solc version: 0.8.24         ·  Optimizer enabled: true  ·  Runs: 200  ·  Block
·····································|··························|·············|········
|  Methods
···············|·····················|·············|·············|·············|········
|  Contract    ·  Method             ·  Min        ·  Max        ·  Avg        ·  # call
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  commitSecretCode   ·      58861  ·      58873  ·      58870  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  createGame         ·     171673  ·     198437  ·     177621  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  dispute            ·          -  ·          -  ·      85347  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  dontDispute        ·          -  ·          -  ·      66939  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  giveFeedback       ·          -  ·          -  ·      59794  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  joinGame           ·          -  ·          -  ·      59285  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  makeGuess          ·          -  ·          -  ·      64487  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  revealCode         ·          -  ·          -  ·      66203  ·
···············|·····················|·············|·············|·············|········
|  Mastermind  ·  startGame          ·          -  ·          -  ·      36993  ·
···············|·····················|·············|·············|·············|········
|  Deployments                       ·                                         ·  % of l
·····································|·············|·············|·············|········
|  Mastermind                        ·          -  ·          -  ·    4059825  ·       1
```

```
·-----------------------------------|-------------|-------------|-------------|--------

9 passing (599ms)
```