COMP0143: Cryptocurrencies

Coursework

University College London

Released: November 8, 2024

Due: December 11, 2024 at 16:00 UK time

Instructions

1. This assignment is part of the mandatory assessment of the *COMP0143: Cryptocurrencies* module and will count **40%** towards your final overall mark.
2. Assignment submission is due via Moodle through the TurnItIn interface on **December 11, 2024 at 16:00 UK time**. Late submissions will be accepted with deductions according to UCL's late submission policy.
3. Only **PDF submissions** that are typeset with LaTeX, *e.g.*, via `https://www.overleaf.com/edu/ucl`, will be accepted. Submissions **must not include screenshots**, *e.g.*, of handwritten or drawn solutions, unless explicitly permitted. Students with disability accommodations are excluded from this requirement.
4. Submissions containing the student's name or other identifying information will have 10 points deducted.
5. Next to the PDF with your answers, you may be asked to hand in additional files, *e.g.*, containing source code, which should be submitted separately to the PDF. In particular, **do not** hand in your files via a zip archive. For more details, please refer to the instructions in the question text.
6. This assignment is open note, open book, and open course resources. You must identify sources as accurately and fully as possible. UCL plagiarism policies will be *strictly* enforced. For more details, see `http://www.ucl.ac.uk/current-students/guidelines/plagiarism`.
7. You are not allowed to consult other people (outside of course staff) on this work. Each student has to work on the assignment individually.
8. Your answers will be judged in terms of their quality, the depth of understanding, and also their brevity. Explain your answers clearly, but succinctly. **Points may be subtracted for answers that are unnecessarily long**. Partial credit may be awarded.
9. The assignment has a maximum of **100 marks** allocated as follows:

	Q1	Q2	Q3	Q4	Total
Marks	25	25	30	20	100 marks

Question 1 [25 marks]

(a) Consider an m-ary Merkle tree, *i.e.*, a hash tree where every internal node has up to $m \geq 2$ children, over a set of elements $S = \{t_1, \ldots, t_n\}$. The hash value for any of these internal nodes is computed as the hash of the concatenation of all of its children.

(i) Suppose $n = 9$ and $m = 3$. How can Alice compute a commitment C to S using such a ternary Merkle tree? Assuming Bob knows C, how can Alice later prove to Bob that t_4 is in S? Justify your answers. **[3 marks]**

(ii) What is the length of the proof l that some value t_i is in S as a function of parameters n and m? **[4 marks]**

(iii) For large n, is it better to use a binary or ternary tree if our goal is to minimize the proof size? Justify your answer. **[5 marks]**

(b) Why do Bitcoin blocks include the root hash of a Merkle tree over all transactions in addition to the full list of transactions? What would be the consequence if this root hash was not included? **[3 marks]**

(c) In class we have discussed the differences between privacy and anonymity and how coins are architected towards these goals (or against them). Consider a privacy coin not discussed in class taken from `https://coinmarketcap.com/view/privacy/`.

(i) Name the coin and explain why you are considering it. Based on the coin's code and/or whitepaper, discuss the privacy and/or anonymity goal(s) and briefly describe the mechanism how it is achieved. **[4 marks]**

(ii) Briefly discuss whether or not the mechanism actually fulfills the expected privacy and/or anonymity goal(s). **[3 marks]**

(iii) Argue if there are any engineering or non-engineering design choices that could help to achieve these goals. **[3 marks]**

Question 2 [25 marks]

(a) Assume Alice and Bob are competing Bitcoin miners. If they discover blocks nearly simultaneously, neither of them will have time to hear about the other's block before broadcasting their own. What determines whose block will end up in the blockchain permanently? Describe the process. **[4 marks]**

(b) How can a miner establish an identity in a way that is hard to fake and connect it to a real-world entity (*e.g.*, the company operating the miner) so that anyone can tell which blocks were discovered by that particular miner? **[4 marks]**

(c) If a miner misbehaves, can other miners boycott the offender on an ongoing basis? If so, how? If not, why? **[4 marks]**

(d) In Bitcoin, the difficulty adjustment algorithm uses timestamps that miners place into blocks to measure the amount of real-world time used to add a batch of 2016 blocks to the longest chain, and then adjust accordingly the proof-of-work difficulty for the next 2016 blocks.

(i) Suppose miners were free to put whatever timestamps they wanted into blocks. How could miners use this power to manipulate Bitcoin's difficulty adjustment algorithm and boost their rewards? Give a concrete example of an attack. **[5 marks]**

(ii) Explain the rules that govern block timestamps in the Bitcoin protocol. Cite the source(s) you used to answer this question; just URLs are fine. **[4 marks]**

(iii) Discuss to what extent the rules in (ii) mitigate the attack(s) that you described in (i). **[4 marks]**

Question 3 [30 marks]

Alice is planning a backpacking trip and worries that her mobile phone with her cryptocurrency wallet keys might get stolen. She investigates possible solutions to keep her funds safe.

(a) Alice comes up with the idea to store her bitcoins in such a way that they can be redeemed by proving knowledge of a password. Accordingly, she stores them in the following `ScriptPubKey` address:

```
OP_SHA256
ff03fc3d437f6e4f4e492e52394f447cc5cdbc499d765d57e7c633457b630068
OP_EQUAL
```

Write a `ScriptSig` script that will successfully redeem the above transaction given the password `bitcoins` and show the contents of the stack after each operation of the script. **[5 marks]**

(b) Assume Alice chooses an 8-digit pin. Explain why her bitcoins can be stolen soon after the UTXOs are posted to the blockchain. **[3 marks]**

(c) Suppose Alice chooses a diceware-generated passphrase consisting of 10 words (for more details on diceware passphrases see `https://en.wikipedia.org/wiki/Diceware`). Is the `ScriptPubKey` above then a secure way to protect her bitcoins? Why or why not? **[6 marks]**

(d) Can you design another passphrase-based mechanism allowing Alice to manage her bitcoins securely? Explain briefly how your system works and what its potential advantages are in comparison to the above `ScriptPubKey` procedure. Does your system have any disadvantages? **[8 marks]**

(e) Alice's best friend Claire, a cryptocurrency enthusiast, recommends Alice to use a 2-of-3 multisig wallet and offers to help her set things up. Briefly describe the functionality of such a wallet and its advantages and how a secure setup could look like. Provide `ScriptPubKey` and `ScriptSig` script entries to access Alice's funds. **[8 marks]**

Question 4 Solidity - Tic Tac Toe [20 marks]

You are given a decentralized application for a tic-tac-toe game with a backend written in Solidity. The game is not yet functional and your task is to complete it. Please read the following instructions carefully.

Docker. To keep setup overheads to a minimum, the entire app is packaged inside a Docker container. To get ready, please install the Docker desktop app on your machine. It may be helpful to go through the Docker tutorial to get familiar with the tool even though it is not strictly necessary to finish this coursework as all instructions provided here are self-contained. After installing the Docker desktop app, you can start the tutorial container via

```
docker run -d -p 80:80 docker/getting-started
```

To view the tutorial browse then to `http://localhost:80`. The time required to finish it is about 2h.

MetaMask. The other tool that you need to interact with your decentralized application is the wallet browser extension MetaMask. Since tic-tac-toe is a two player game you need two installations of MetaMask, *e.g.*, in two different browsers and Chrome, Firefox, and Brave should all work. You can also run it in one browser by creating two different users and by installing MetaMask for each. Afterwards you will need to configure MetaMask to connect to your local test chain. To do so, first add a new network in Metamask with the following details:

- Network name: `Localhost 8545`
- RPC URL: `http://localhost:8545`
- Chain ID: `1337`
- Currency symbol: `ETH`

Then make sure you have switched to this new network in MetaMask. At this point it's not functional yet because you have to install the local test chain first, see the instructions further below.

Running the Game. The smart contract you should complete for this assignment is located at `contracts/TicTacToe.sol`. The missing code snippets are marked with `/*Please complete the code here.*/`. In total there are **7 functions** that you need to complete:

- `_threeInALine()` **[3 marks]**
- `_getStatus()` **[3 marks]**
- `_checkStatus()` **[2 marks]**
- `myTurn()` **[3 marks]**
- `_myTurn()` **[3 marks]**
- `validMove()` **[3 marks]**
- `_validMove()` **[3 marks]**

To set up and play your tic-tac-toe game, you need to:

1. Install and start the local Ganache test chain:

   ```
   docker run --name ganache -p 8545:8545 -d trufflesuite/ganache-cli:latest -g 0
   ```

2. Build the tic-tac-toe game:

   ```
   docker build -t tic-tac-toe .
   ```

   Ensure you run this command from the directory containing the Dockerfile for the project. *Note:* This command will fail the first time you execute it because the smart contract is not yet complete and still has some syntax errors. Hence, your first task is to fix these syntax errors after which you can start the app. For testing purposes you can temporarily comment out the line `RUN npx hardhat compile` in the `Dockerfile` which omits compiling the smart contract. Make sure to re-include it later on otherwise your app will not work!

3. Start the web server:

```
docker run -p 8080:8080 -d tic-tac-toe
```

4. Browse to `http://localhost:8080/` in two separate web browsers, each with its own MetaMask installation and connected to the local test chain. Follow the instructions on the site to start the game. Note that after making a move, it may take some time until it appears on the board in the other browser which is due to the sync delay of the web frontend.

5. To stop the local Ganache test chain:

```
docker stop ganache
```

6. To remove all Docker containers:

```
docker container prune
```

Testing and Debugging.

- We provide several test cases to test the basic functions of your contract, which, however, is not complete as you are supposed to test your application through the UI and with MetaMask.
- To check the (incomplete) test results, you can run

  ```
  docker run tic-tac-toe npm test
  ```

  If you like, you are encouraged to extend the tests yourself.
- Our grading script will follow a similar methodology as the test script, *i.e.*, to test your smart contract and grade based on the test results.
- You can view the output and error messages of MetaMask using the browser developer tools (*e.g.*, Inspect > Console in Brave/Chrome).
- After rebuilding your app and reloading the web UI, MetaMask sometimes complains about invalid nonces. To resolve this issue you can delete the transaction history by resetting MetaMask via Account > Settings > Advanced > Reset Account.
- If you are located in China, make sure to connect to the UCL network via a VPN before going through the setup otherwise some steps may fail (like installing packages via `npm` inside Docker containers).

Submission. As your solution, please only submit the completed `TicTacToe.sol` file.