UNSW COMP6451
Assignment 2 : Ethereum Programming
Total Marks: 35
Due Date: 5pm, April 11, 2022

NeverPay is a startup Fintech company based on the latest hot new business model in consumer payments, which enables consumers to buy expensive goods without ever having to pay for them. (The catch is that they have to make interest payments to NeverPay for the rest of their lives. Really, this is just what banks have been calling "interest-only loans" for a long time, but hey, wrap it in some tech and put "Pay" in the company name and you have a startup sensation!)

To further show how innovative they area, NeverPay has decided to conduct an equity fundraising on the Ethereum blockchain, allowing investors to buy NeverPay shares using cryptocurrency, and representing shareholdings on Ethereum rather than an old-tech web platform like ComputerShare.

You have been commissioned to develop a smart contract for NeverPay's sale of shares. The client has provided the following specifications:

1. A total of 10,000 shares are to be sold.
2. The minimum price per share is 1 Ether, any bid to buy shares for less than this amount will not be issued shares.
3. Investors should be identified by means of an anonymous Ethereum address - the system should not store personal information on the blockchain.
4. In order to encourage competition amongst the investors and maximize the amount of money raised, the share sale is to be conducted via a blind auction, consisting of two rounds.
5. Deadlines for the rounds are set as (Round1: April 20, 2022, Round2: April 27, 2022) at the time the smart contract is created. Actions that were supposed to have been taken in a round should be rejected after the round deadline.

6. In the first round, investors wishing to buy shares should submit a blinded commitment of an offer to buy a given number of shares at a given price (in Ether) per share. While the first round is running, it should not be possible for any investor to determine from public information on the blockchain what number of shares any other investor has offered to buy, nor what price they have offered to pay.

7. An investor may submit multiple bids in the first round. Any bid may be withdrawn until the first round deadline.

8. In the second round, the investors are required to open their commitments, revealing the number of shares in the bid and the price offered. It should not be possible for the investor to cheat in this round, revealing information that differs from what they committed to in the first round.

9. In this round, the investor is also required to submit, to the smart contract, their payment in full (number of shares $\times$ price per share) in Ether for the shares that they are offering to buy.

10. After the deadline for the end of the second round has passed, any commitment that has not been opened in the second round, or which has been been opened but for which the correct payment has not been provided, should be treated as an invalid, unsuccessful bid. The remaining bids are treated as valid and potentially successful (but may still be unsuccessful because the offer was oversubscribed, and there is more demand for shares than are being sold).

    Shares should then be issued to the investors submitting the valid bids with the highest prices. Suppose the valid bids are $(i_1, s_1, p_1), \ldots, (i_n, s_n, p_n)$, where each i_i is an investor, each s_i is a number of shares, and each p_i is a price, with $p_1 \geq p_2 \geq p_n$. Then the 10,000 shares should be issued to the investors so that i_1 gets s_1 shares, i_2 gets s_2 shares, and so on, until all 10,000 shares have been issued. The final investor i_k receiving shares may receive only some of the shares in their bid. If they receive s shares, then they pay $s * p_k$ for these shares, and they should have the remainder $(s_k - s) * p_k$ of their payment refunded. The bid of every investor $i_{k+1} \ldots i_n$ after this final successful investor i_k is treated as an unsuccessful bid.

11. Ties between bids making the same price offer should be handled in the order in which the bids were submitted during round 1.

12. Any unsuccessful bids should have payments that have been made to the smart contract refunded to the bidder.

13. After shares have been issued, it should be possible for investors to transfer their shares to another investor, using ERC-20 standard operations.

14. The smart contract should be cost effective for the NeverPay to run. To the largest extent possible, transaction costs should be borne by the investors rather than by NeverPay.

15. The smart contract should also treat the investors uniformly with respect to the costs imposed on them to interact with the smart contract. Requiring one investor to pay the transaction costs for computation that relates to another investor's shares should be avoided.

As stretch goal for the assignment, consider the following issue. There is a risk that the ASIC (the Australian Securities and Investment Commission, which regulates investment products) will declare NeverPay's share sale is an illegal unregistered Managed Investment Scheme, which could result in NeverPay being fined and forced to return the money raised. To avoid this classification and for the share sale to be compliant with ASIC regulations, NeverPay will need to ensure that all investors to which it sells shares are *sophisticated investors*. The requirements for being a sophisticated investor are to have an annual income over $250,000 or a net worth of over $2.5 million.

To assist companies raising funds on the blockchain to comply with this requirement, ASIC has proposed that certain trusted organizations will be given authorization by ASIC to issue certificates attesting that the owner of a given Ethereum address is a sophisticated investor. (For example, banks and the Australian Post Office may be authorized to issue such certificates, based on ASIC's trust that these organizations can be relied upon to check investor information for compliance before issuing such a certiticate.)

To enable smart contracts to check that an Ethereum address is owned by a sophisticated investor, ASIC wishes to create a SophisticatedInvestorCertificateAuthorityRegistry smart contract on the Ethereum blockchain that acts as a registry of the public keys of organizations that have been authorized to issue "sophisticated investor certificates". The functionality of this smart contract should be as follows:

- The smart contract should store a set of Ethereum public keys to be used as signature verification keys.
- Only ASIC should be able to add or delete public keys to this set.
- There should be a function that can be called by other smart contracts to check that a given public key is currently in the set of authorized keys.

The certificate authorities themselves do not wish to pay gas costs and they are not expected to operate an Ethereum smart contract. Instead, they use their private signature keys to sign messages that represent the following statement:

> "The owner of Ethereum address `<address>` is a sophisticated investor for year `<year>`."

(The precise representation of this information is for you to design.) They provide a copy of the signed message, together with the authority's public signature verification key to the investor (e.g., by email to the investor).

When the investor wishes to make an investment that requires them to prove their sophisticated investor status, they can provide a copy of the certificate issued by the authority, together with proof that they actually control the address in the certificate.

The stretch goal for the project is to

1. Implement the SophisticatedInvestorCertificateAuthorityRegistry smart contract.
2. Write code that a certificate authority can use to create sophisticated investor certificates.
3. Extend the NeverPay smart contract so that investors can use certificates issued by authorized certificate authorities to prove their sophisticated investor status to NeverPay. The result should be that NeverPay only issues shares to sophisticated investors.

(Hint: The Ethereum function **ecrecover** and frameworks such as Web3.py or Web3.js are relevant to this part. You may also wish to investigate proposals such as EIP-721 and EIP-2612.)

Deliverables

The client has specified the following deliverables for this project. Each deliverable is associated with a payment (in marks), subject to fully satisfactory performance. The project has a strict deadline, so you should attempt the following basic goals first:

1. (8 marks) A report (pdf format) describing your design and implementation of the overall system. The report should
   - Describe the data model, and how you have chosen to implement this design using Solidity data structures.
   - In case use of the smart contract requires off-chain computations not implemented in the smart contract, explain what this code does and how to compile and/or operate it. You are not required in this assignment to develop a fancy user interface for any such code: some simple command-line scripts suffice.
   - For each of the requirements above, briefly indicate where and how your code meets the requirement. Briefly explain each of the functions in your code. In case you identified any missing requirements in the course of your analysis of the application, state what these are and what you have done to implement them.
   - Provide an analysis of the running costs of the application from the point of view of NeverPay and a bidding investor, and how this depends on factors such as the total number of bids. Take into account current information on the costs of transactions on the Ethereum public blockchain, and the gas costs of running your code.
   - Reflectively discuss the overall suitability of the Ethereum platform for this application.

- Explain how your code avoids common Solidity security vulnerabilities like reentrancy attacks.
- Describe any security considerations concerning the system and its operation that you consider should be pointed out to the client.

Proper acknowledgement of any sources of information or libraries you have used in your project is required.

2. (10 marks) A directory **contracts** containing smart contracts in Solidity for your implementation of the basic functionality. Your code should be well documented.

3. (7 marks) A directory **test** containing test cases to validate the correctness of your smart contract implementation. Your report should describe the approach that you have taken to testing, and summarize the test scenarios that you have constructed. It should be possible to execute your tests using the Truffle testing framework running against a Ganache instance of the Ethereum blockchain, and the report should contain all the information that is required to determine how to run your tests.

4. (10 marks) Once you have met the basic requirements, attempt the stretch goal. If you attempt this part, your report should contain a section describing what you have done for this part, you should include test cases for this functionality, and it should be clear from your report how to run these test cases.

Submission

This assignment is required to be your individual work. Submit your project from your CSE account using the command

```
give cs6451 assignment2 <tarfile>
```

on a CSE server.