University of Edinburgh	*Fall 2020-21*
Blockchains & Distributed Ledgers	*Instructor:* Aggelos Kiayias *Teaching Assistant:* Dimitris Karakostas

Assignment #3 (Total points = 30)

Due: Monday 18.1.2021, 16.30

Part 1

In this assignment you will interact with a smart contract that has been deployed on Ethereum's testnet, Ropsten. To get Ropsten Ether you can use a faucet, such as Metamask's faucet. The contract's address is *0x8B9CCE0437c421A53f033734772430A105BbBE89* and its code is available here. You should call the "register" function at least once, with your student number as the second argument. Your report should detail how you managed to perform the transaction, detailing what key you used and how you found it.

Part 2a: Smart Contract Programming Part II - Token

In this assignment you will create your own custom token. Your contract should define the following public API (plus any other functions/variables you deem necessary):

- **tokenPrice**: a uint256 that defines the price of your token in wei; each token can be purchased with *tokenPrice* wei
- ***Purchase(address buyer, uint256 amount)***: an event that contains an address and a uint256
- ***Transfer(address sender, address receiver, uint256 amount)***: an event that contains two addresses and a uint256
- ***Sell(address seller, uint256 amount)***: an event that contains an address and a uint256
- ***Price(uint256 price)***: an event that contains a uint256
- **buyToken(uint256 amount)**: a function via which a user purchases *amount* number of tokens by paying the equivalent price in wei; if the purchase is successful, the function returns a boolean value (*true)* and emits an event *Purchase* with the buyer's address and the purchased amount
- **transfer(address recipient, uint256 amount)**: a function that transfers *amount* number of tokens from the account of the transaction's sender to the *recipient*; if the transfer is successful, the function returns a boolean value (*true)* and emits an event *Transfer,* with the sender's and receiver's addresses and the transferred amount
- **sellToken(uint256 amount)**: a function via which a user sells *amount* number of tokens and receives from the contract *tokenPrice* wei for each sold token; if the sell is successful, the sold tokens are destroyed, the function returns a boolean value (*true)* and emits an event *Sell* with the seller's address and the sold amount of tokens
- **changePrice(uint256 price)**: a function via which the contract's creator can change the *tokenPrice*; if the action is successful, the function returns a boolean value (*true)* and emits an event *Price* with the new price (Note: make sure that, whenever the price changes, the contract's funds suffice so that *all tokens* can be sold for the updated price)
- **getBalance()**: a view that returns the amount of tokens that the user owns

Your contract should use OpenZeppelin's "safe math" library. Specifically, this is the only library that you can use, but you do have to use it. Also, copy/pasting the library's functions in your own contract is not acceptable; instead you use a deployed instance of the library.

You should implement the smart contract and deploy it on Ropsten. After deploying your contract, you should buy, transfer, and sell a token in the contract that one of your fellow students created. Also, after at least one token on your contract has been bought, you should double your token's price.

You report should contain:

- A detailed description of your high-level design decisions, including (but not limited to):
    - What internal variables did you use?
    - What is the process of buying/selling tokens and changing the price?
    - How can users access their token balance?
    - How did you link the "safe math" library to your contract?
- A detailed gas evaluation of your implementation, including:
    - The cost of deploying and interacting with your contract.
    - Techniques to make your contract more cost effective.
- A thorough listing of potential hazards and vulnerabilities that can occur in the smart contract. Provide a detailed analysis of the security mechanisms that can mitigate these hazards.
- The transaction history of the deployment of your contract and your interaction with your fellow student's contract.
- The code of your contract.

Part 2b: KYC Considerations and Token issuance

Suppose that the smart contract issuer has to do that to comply with regulation related to KYC ("know your customer"), where token issuance can happen only in case some identification document has been provided. Therefore, you want to associate with each buyToken operation an encrypted file, that corresponds to an identification document of the token recipient.

Describe in your report a way you can use a public-key encryption scheme to incorporate such a ciphertext in the buyToken operation, as well as any other changes needed in the above API. Is it possible to implement this process completely on-chain? Describe in detail the steps and tools (using any relevant material from the lectures) needed to implement this.