University of Edinburgh	*Fall 2021-22*
Blockchains & Distributed Ledgers	

Assignment #3 (Total points = 100)

Due: Monday 10.1.2022, 16.30

You should submit a single PDF report that contains your answers for all of the following parts.

Part 1 (10 points)

In this part, 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 *0xde3a17573B0128da962698917B17079f2aAbebea* 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 (70 points)

In this assignment you will create your own custom token. Your contract should follow the below public API (plus any other *private/internal* 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

You should implement the smart contract and deploy it on Ropsten. Your contract should be as secure and gas efficient as possible. After deploying your contract, you should buy, transfer, and sell a token in the contract. After at least one token on your contract has been bought, you should double your token's price.

Also, a custom library has been deployed on Ropsten; you can find its source code here and its address is *0xc0b843678E1E73c090De725Ee1Af6a9F728E2C47*. Whenever your contract tries to make payment to an account, it should use the library's *"customSend"* function. **Note:** Your contract should be linked to and use *the deployed instance* of the library; you *should not* deploy your own instance of the library alongside your contract, nor copy/paste its code in your contract.

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 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.
  - What was the gas impact of using the deployed library instance, compared to including its code in your contract?
- A thorough listing of potential hazards and vulnerabilities that can occur in the smart contract and a detailed analysis of the security mechanisms that can mitigate these hazards.
- The transaction history of the deployment of and interaction with your contract.
- The code of your contract.

Part 2b: KYC Considerations and Token issuance (20 points)

Suppose that the smart contract issuer has 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.