University of Edinburgh	*Fall 2022-23*
Blockchains & Distributed Ledgers	

Assignment #3 (Total points = 100)

Due: Monday 9.1.2023, 12.00 (noon)

Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page: https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Part 1: Smart Contract Programming Part II - Token (70 points)

In this assignment you will create your own custom token. Your contract should implement the below public API:

- **owner**: a public payable address that defines the contract's "owner", that is the user that deploys the contract

- ***Transfer(address indexed from, address indexed to, uint256 value)***
- ***Mint(address indexed to, uint256 value)***
- ***Sell(address indexed from, uint256 value)***

- **totalSupply()**: a view function that returns a uint256 of the total amount of minted tokens
- **balanceOf(address _account)**: a view function returns a uint256 of the amount of tokens an address owns
- **getName()**: a view function that returns a string with the token's name
- **getSymbol()**: a view function that returns a string with the token's symbol
- **getPrice()**: a view function that returns a uint128 with the token's price (at which users can redeem their tokens)
- **transfer(address to, uint256 value)**: a function that transfers *value* amount of tokens between the caller's address and the address *to*; if the transfer completes successfully, the function emits an event *Transfer* with the sender's and receiver's addresses and the amount of transferred tokens and returns a boolean value (*true)*
- **mint(address to, uint256 value)**: a function that enables *only the owner* to create *value* new tokens and give them to address *to*; if the operation completes successfully, the function emits an event *Mint* with the receiver's address and the amount of minted tokens and returns a boolean value (*true)*
- **sell(uint256 value)**: a function that enables a user to sell tokens for wei at a price of *600 wei per token*; if the operation completes successfully, the sold tokens are removed from the circulating supply, and the function emits an event *Sell* with the seller's address and the amount of sold tokens and returns a boolean value (*true)*
- **close()**: a function that enables *only the owner* to destroy the contract; the contract's balance in wei, at the moment of destruction, should be transferred to the *owner* address

- **fallback** functions that enable anyone to send Ether to the contract's account
- **constructor** function that initializes the contract as needed

You should implement the smart contract and deploy it on Ethereum's testnet, Goerli. 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.

Your contract should **implement the above API exactly**. *Do not* omit implementing one of the above variables/functions/events, *do not* change their name or parameters, and *do not* add other public variables/functions. You can define other private/internal functions/variables, if necessary.

Your 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?
  - 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 list 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 as an Appendix to your report.
- The code of your contract as an Appendix to your report. *(Note: The contract should be both at the end of your PDF report and submitted as a separate file. See submission instructions below.)*

Part 2: Using libraries (10 points)

A custom library has been deployed on Goerli. Its source code is here and its address is *0x9DA4c8B1918BA29eBA145Ee3616BCDFcFAA2FC51*.

You should adapt your contract from part 1, such that whenever it tries to send wei to an account, it should use the library's *"customSend"* function instead of call/send/transfer. You can reuse the contract from part 1 and make only the necessary changes (in other words, there is no need to make an entirely new implementation for this part).

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 "customSend" as a function of your contract and have the contract run that code (instead of calling the deployed library instance).

In your report:

- Write a small description of how you linked your contract to the deployed library.
- Provide the code of your contract and any other relevant deployment information as an Appendix to your report. *(Note: The contract should be both at the end of your PDF report and submitted as a separate file. See submission instructions below.)*

Part 3: KYC Considerations and Token issuance (20 points)

Suppose that the issuer of the smart contract from Part 2 has to comply with regulation related to KYC ("know your customer"). Specifically, token issuance can happen only in case some identification document has been provided. Therefore, you want to associate each *Mint* operation with a file that corresponds to an identification document of the tokens' recipient. In addition, the recipient's ID should be kept as private as possible; ideally, nobody beyond the contract's issuer should gain any information about the document's content.

Describe in your report a way to use a public-key encryption scheme to incorporate such a document in the contract's operation. Detail any changes needed in the above API (you don't need to actually implement them). Is it possible to conduct this process completely on-chain? Describe in detail the steps and tools (using any relevant material from the lectures) needed to implement this functionality.

Submission

You should submit **three files** via Learn (all in the same Learn submission).

First, a solidity file that contains the code of your smart contract for part 1. The name of the file should be your exam number, followed by "_part1" (e.g., *B123456_part1.sol*).

Second, a solidity file that contains the code of your smart contract for part 2. The name of the file should be your exam number, followed by "_part2" (e.g., *B123456_part2.sol*).

Third, a PDF report that covers the questions outlined above for parts 1-3. The report, excluding the Appendices detailed above, should be at most 10 pages (font size at least 11, margin at least 1 inch all around). The name of the file should be your exam number (e.g., *B123456.pdf*).