FUNDAMENTOS Y APLICACIONES DE BLOCKCHAINS

Homework 3

Depto. de Computación, UBA, 2do. Cuatrimestre 2025 *9/10/25*
Student: Juan DElia *Due: 21/10/25, 15:00 hs*

Instructions

- Upload your solution to Campus; make sure it's only one file, and clearly write your name on the first page. Name the file '<your last name>_HW3.pdf.'

  If you are proficient with LaTeX, you may also typeset your submission and submit in PDF format. To do so, uncomment the "%\begin{solution}" and "%\end{solution}" lines and write your solution between those two command lines.

- Your solutions will be graded on *correctness* and *clarity*. You should only submit work that you believe to be correct.

- You may collaborate with others on this problem set. However, you must **write up your own solutions** and **list your collaborators and any external sources (including ChatGPT and similar generative AI chatbots)** for each problem. Be ready to explain your solutions orally to a member of the course staff if asked.

This homework contains 4 questions, for a total of 65 points.

1. **Bitcoin transactions:**

   (a) (7 points) Describe the mechanism used in the actual Bitcoin application that the miners follow in order to insert transactions into a block. Would the Consistency (aka Persistence) and Liveness properties we saw in class, as well as the V, I, R functions have to be modified to capture the actual mechanism? Elaborate.

Solution: Para describir el mecanismo que usan los mineros de Bitcoin para agregar un bloque use los siguientes recursos:

En la seccion 5 del paper

El sitio de developers de Bitcoin

El mecanismo se resume en:

1. Nuevas transacciones son transmitidas en la red a los nodos.
2. Cada nodo recolecta transacciones para formar un bloque, sumando la public key a donde se debe enviar el coinbase.
3. Cada nodo intenta resolver el PoW para el target correspondiente
4. Cuando un nodo encuentra una PoW transmite el bloque en la red para todos los nodos
5. Los nodos de la red aceptan el bloque si todas las transacciones son validas usando las merkle root y verificando que no haya double spending
6. Los nodos expresan su aceptacion de un bloque poniendose a crear el siguiente bloque de la cadena, usando como "hash del bloque anterior" el hash del bloque que aceptan

No es necesario cambiar Consistency (Everyone sees the same history) ni Liveness (Everyone can add new transactions). Ambas tienen sentido en el contexto del mecanismo real de bitcoin. Por como funciona el mecanismo la idea es que mediante PoW se mantenga la persistencia de las transacciones, ya que se extienden las cadenas donde se usa "esfuerzo" si la mayoria de los nodos son honestos todos van a ver la misma cadena, o al menos el mismo prefijo (los ultimos bloques son mas volatiles).

Tampoco hace falta cambiar Liveness (Everyone can add new transactions). Cualqueira puede emitir transacciones en la red.

No nos interesa cambiar la idea de las funciones. Por como estan definidos en el paper seccion 3.1 sirven para expresar la esencia del mecanismo:

- V: The content validation predicate receives as input the content of a chain C, denoted by xC, and will return 1 if and only if the contents are consistent with the intended application implemented on top of the chain.

Esto nos intteresa hacerlo, lo hacen los nodos de la red cuando verifican que todas la transacciones de la cadena sean validas con las merklee root y que no haya double spending.

- I: It receives as input a tuple, (st, C, round, Input(), Receive()), that stands respectively for state data st, current chain C, current round round, contents of input tape Input() and contents of network tape Receive(). Given these, it will produce an updated state st' as well as an input x that should be the next input to be inserted in a block. For instance, I(·) can be as simple as copying the contents of the input tape into x and keeping st = ϵ , or performing a more complex operation that involves parsing C or even maintaining old input values that have not yet been processed as part of the state st.

- R: It receives as input a chain C and provides an interpretation of it. In the simplest case it can be just returning xC and leaving it to the callee to process the contents of the chain.

  Nos interesa, en la aplicacion de blockchain cumple el proposito de listar las transacciones de los bloques.

(b) (3 points) Describe the purpose of a *coinbase* transaction.

Solution: (Referencia whitepaper de Nakamoto 6.Incentive)

La transaccion coinbase es la primera de cada bloque y representa la recompensa que se le va a pagar a quien mine el bloque. Esta transaccion crea bitcoins, sirve no solo como recompensa sino como forma de poner moneda en circulacion, ya que no hay una autoridad central que lo haga.

2. **Proofs of work:**
   (a) (5 points) Describe the purpose and implementation of the **2x1 PoW** technique.
   (b) (7 points) Let T_1 and T_2 be the target values in 2x1 PoW, and κ the size of the hash function's output. What relation should they satisfy for the technique to work? Elaborate.
   (c) (8 points) Design and argue correctness of an ℓx1 PoW scheme. Note that such a scheme would enable an ℓ-parallel blockchain. What properties of the blockchain or ledger application would, if any, benefit from a parallel blockchain? Elaborate.

3. **Strong consensus:** We saw how the Bitcoin backbone protocol can be used to solve the *consensus* problem (aka *Byzantine agreement*). In the *strong consensus* problem, the *Validity* condition is strengthened to require that the output value be one of the honest parties' inputs—this property is called *Strong Validity*. (Note that this distinction is relevant only in the case of non-binary inputs.)
   (a) (5 points) What should be the assumption on the adversarial computational power (similar to the "Honest Majority Assumption") for Strong Validity to hold?
   (b) (5 points) State and prove the Strong Validity lemma.

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

    - **<u>owner:</u>** 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):*** an event that contains two addresses and a uint256
    - ***Mint(address indexed to, uint256 value):*** an event that contains an address and a uint256
    - ***Sell(address indexed from, uint256 value):*** an event that contains an address and a uint256
    - **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's 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 the course's Sepolia Testnet. 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 as specified**. *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.

You should provide:

(a) (5 points) 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?

(b) (5 points) 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.

(c) (5 points) 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.

(d) (5 points) The transaction history of the deployment of and interaction with your contract.

(e) (5 points) The code of your contract.