Fundamentos y Aplicaciones de Blockchains

Homework 3

Depto. de Computación, UBA, 2do. Cuatrimestre 2025 *9/10/25*

Student: Felipe Luc Pasquet *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: Los mineros agrupan transacciones validas y forman bloques que intentan agregar a la blockchain. Un bloque es una secuencia de transacciones $x = tx_1, \ldots, tx_f$.
Las transacciones tx tienen la forma "$tx = \{a_1, a_2, ..., a_i\} \longrightarrow (\sigma, \{(a'_1, b'_1), \ldots, (a'_o, b'_o)\})$," donde las $a_1, ..., a_i$ son las cuentas (inputs) de las que se debita, $a'_1, ..., a'_o$ son las direcciones en las que se acreditan los fondos (outputs) y $b'_1, ..., b'_o$ es la cantidad de fondos que se le transfiere a esa dirección. σ es un vector $\langle (vk_1, \sigma_1), \ldots, (vk_i, \sigma_i) \rangle$ que contiene la clave de verificación y la firma de cada dirección a_j.

Además de que sean validas las firmas de cada cuenta, es necesario que las transacciones sean validas con respecto a la blockchain $X = \langle x_1, \ldots, x_m \rangle$, es decir que $\sum_{j=1}^{i} b_j \geq \sum_{j=1} b'^o_j$, donde b_j es el saldo de la cuenta a_j en la última transacción que involucró a a_j en X.

Si $\sum_{j=1}^{i} b_j > \sum_{j=1} b'^o_j$, osea si el valor de los inputs es mayor al valor de los outputs de una transacción, esa diferencia es un "fee/comisión" que se puede quedar el minero que agrega la transacción a un bloque. Este fee, llamemoslo $e = \sum_{j=1}^{i} b_j - \sum_{j=1} b'^o_j$, se agrega junto al "subsidio" o premio de bitcoins que son creados en ese bloque. Esta recompensa puede ser reclamada por el minero en una transacción especial que se llama **coinbase**, la cual es la primer transacción de todas en el bloque que crea el minero.

Tendríamos que modificar la función V para que cumpla todo lo que dijimos recién y la función I para que genere también la transacción **coinbase**, que es necesaria para que el bloque sea valido.

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

Solution: Como dijimos en el punto anterior, **coinbase** es el nombre de la primer transacción de un bloque. Es la manera de que el minero pueda reclamar la recompensa de minar un bloque *(las monedas nuevas que se crean + los fees de las transacciones que agregó al bloque)*.
Esta transacción especial tiene un único input vacío, y la suma de los outputs no puede superar a $subsidy + \sum_{i=1}^{f} e_i$ (la suma de todos los fees de las

transacciones más el subsidy, que son los nuevos bitcoins que se crean).

Para este ejercicio utilicé información de
`https://learnmeabitcoin.com/technical/mining/coinbase-transaction/`
y de la sección 6.2 de `https://eprint.iacr.org/2014/765.pdf`

2. **Proofs of work:**

(a) (5 points) Describe the purpose and implementation of the **2x1 PoW** technique.

Solution: El proposito es que si se están ejecutando 2 protocolos simultaneamente, no tener que realizar 2 POW distintas, ya que esto acarrea ciertos problemas, combinando los protocolos y realizando 1 solo POW. En vez de que los dos protocolos usen distintas funciones de hash H_1 y H_2, hashean los datos una sola vez con una sola funcion H, y esto permite que el q también sea el mismo para los dos protocolos.

La modificación básicamente consiste en cambiar la estructura de las POWs de duplas de la forma (w, ctr), a tuplas de $(w, ctr, label)$, donde label es un k-bit string neutral desde el punto de vista del pow. Luego hay que modificar los algoritmos de verificación del POW de los protocolos:

Algorithm 7 The *double proof of work* function, parameterized by q, T and hash functions $H(\cdot), G(\cdot)$ that substitutes steps 2-11 of two POW-based protocols.

function double-pow(w_0, w_1)
  $B_0, B_1 \leftarrow \varepsilon$
  $ctr \leftarrow 1$
  $h \leftarrow \langle G(w_0), G(w_1) \rangle$
  while $(ctr \leq q)$ **do**
    $u \leftarrow H(ctr, h)$
    if $(u < T) \wedge (B_0 = \varepsilon)$ **then**
      $B_0 \leftarrow \langle w_0, ctr, G(w_1) \rangle$
    end if
    if $([u]^{\mathsf{R}} < T) \wedge (B_1 = \varepsilon)$ **then**
      $B_1 \leftarrow \langle w_1, ctr, G(w_0) \rangle$
    end if
    $ctr \leftarrow ctr + 1$
  end while
  return $\langle B_0, B_1 \rangle$
end function

(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.

Solution: Esto la verdad no me quedó del todo claro, porque en el paper usan el mismo T, y en la clase si bien usan dos T distintos no me quedó claro

cual tiene que ser la relación.

(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?

   Solution: Si el input no es binario, es decir si el input viene de un conjunto arbitrario V, $|V| > 2$, la cota que hay que imponer al poder de hashing de los adversarios es que no sea mayor a $(1 - \delta)/|V|$. Esto nos asegura que la proporción de los bloques de la blockchain controlados por el adversario son menos que $\frac{1}{|V|}$.

   (b) (5 points) State and prove the Strong Validity lemma.

   Solution: Lema: Si el poder de hash del adversario está acotado por $\frac{(1-\delta)}{|V|}$, entonces el valor de salida del protocolo de consenso será uno de los valores de entrada de los participantes honestos.
   Demostración: La prueba se basa en la propiedad de calidad de la cadena (Chain Quality). Con este límite más estricto sobre el adversario, se puede garantizar que la proporción de bloques adversarios en la cadena es menor que $\frac{1}{|V|}$, entonces es menor que $\frac{1}{|V|-1}$. Esto asegura que, incluso en el peor de los casos donde los votos de los honestos se dividen entre $|V| - 1$ opciones, el adversario no tiene suficientes bloques para crear una pluralidad para un valor que ningún honesto propuso.

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

   - **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):*** 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?

Solution:
Todas las variables que usé son:
Un diccionario para almacenar el balance de la cuentas, un string para el nombre del token (LukaToken) y un string para el symbol (luKa). También hay una variable address que guarda el owner del contrato, una variable uint256 para la cantidad total de tokens (_totalsupply), otra para la cantidad de decimales que se pueden usar del tokan (_decimals) y otra para el precio de venta del token (price).
Los tokens no se pueden comprar (dado que no especificaron que hagamos la función buy), pero se pueden conseguir si el dueño mintea nuevos tokens o si algún poseedor de los tokens los transfiere. Luego una vez que alguien tiene tokens, puede venderlos a un precio de 600wei el token. Como el token está expresado con 2 decimales en realidad todas las cantidades de los token aparecen multiplicadas por 100, y el precio de 0.01 luKa (que es la medida más chica del token luKa) es 6 wei. Sin embargo como no agregué la función decimals esto no funcionó correctamente, así que parece que simplemente hay 100 veces más luKa de lo que quería mintear, y tienen un valor de 6 wei cada uno.
Cualquier usuario puede vender sus tokens siempre y cuando el contrato tenga los fondos suficientes para hacer la transacción. Una vez vendidos, esos tokens son ”destruidos” y la cantidad total de tokens es reducida.
Los usuarios pueden ver su balance de luKa con la funcion getBalance

(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.

Solution: El costo de crear el contrato fue de 1143116 gas, 0.001699435512462527 ETH.
Ver el nombre del token, su apodo, el dueño, el balance de una cuenta, el precio y la cantidad total es gratis.
Hacer una transferencia de una cuenta a otra tiene consumió 53117 gas.
Mintear nuevos tokens consumió 37088 gas.
Vender tokens consumió 44026 gas.
Por último ejecutar la función close() la cual me dio todo el balance de eth del contrato consumió 28600 gas.

(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.

Solution: No hice nada contra el overflow, así que eso podría ser una vulnerabilidad. De hecho debería poder hacerse un ataque de underflow cuando se venden tokens ya que solo verifica si la cantidad indicada para vender es mayor a cero, pero no si es mayor al balance. También debería poder venderse cualquier cantidad siempre que se tenga al menos 1 token en el balance. Sin embargo no logré realizar estos ataques, pues cuando intento hay un error panic por el overflow y la transacción falla.

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

Solution: El contrato del token es 0xda394181672499A52B045063cff01874b25d9146.
Se puede ver el token en la testnet en
`https://sepolia.etherscan.io/token/0xdbf9f10afc9d8c9ba875e49c7d7372a453ee6170`
y se pueden ver todo el historial de transacciones en
`https://sepolia.etherscan.io/address/0xdbf9f10afc9d8c9ba875e49c7d7372a453ee6170`

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

Solution: El código del token se puede leer en
`https://github.com/feliP-P/Blockchains/blob/main/lukatoken.sol`