Fundamentos y Aplicaciones de Blockchains

Homework 2

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

Student: Gaspar Joel Zuker *Due: 7/10/25, 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>_HW1.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 60 points.

1. We saw in class the notion of *digital signatures* and their security properties, *existential unforgeability* being an important one.

   (a) (5 points) Describe the purpose of a *Public-Key Infrastructure* (PKI). Can the security properties of a digital signature be guaranteed without a PKI? Elaborate.

   Solution: Las propiedades de seguridad de una firma digital son: Existenial Unforgability - Una firma digital con esta propiedad no tiene el problema de "Existencial forgery", es decir, no puede ser creado un par mensaje - firma valído para una clave pública si no se posee la clave privada correspondiente. Esta es una propiedad importante para evitar transacciones maliciosas.

   El concepto de public key infrastructure es una parte importante en muchos protocolos de firmas digitales, ya que consiste en tener una entidad confiable que genera los pares de claves públicas y privadas. Este modelo tiene ventajas, ya que permite tener algoritmos mas eficientes para generar consenso.

   Sin embargo, este sistema no es la única manera de mantener las propiedades de seguridad de sus firmas digitales. Aunque en caso de no utilizarse se requiere un porcentaje mayor de nodos honestos para garantizar su funcionamiento, además de aumentar la complejidad de los algoritmos.

   (b) (5 points) Bitcoin transactions use the ECDSA signature scheme. Does Bitcoin assume a PKI? If not, reconcile with the above argument.

   Solution: Confiar en una entidad va en contra de la filosofía de bitcoin y de la blockchain en general, por ende, bitcon no utiliza un trusted setup. En su lugar utiliza un protocolo con caracteristicas diferentes.

   El modelo de bitcoin es "permisionless" (cualquiera puede participar, no se conocen a priori los nodos participantes) y no utiliza una PKI. Utiliza una comunicación de "message diffusion" (con pki se utilizan canales punto a punto).

2. (10 points) Refer to Algorithm 4 (Main Loop) in [GKL15]. We saw in class that blockchains would start from a "genesis" block, which would provide an unpredictable string to start mining from. Yet, notice that in Algorithm 4 mining starts from the empty string ($\mathcal{C} \leftarrow \varepsilon$). How come? Explain why this works.

Solution: El bloque de génesis no es necesario para que funcione el algoritmo de blockchain, es utilizado en la práctica para garantizar que nadie puede comenzar a generar bloques que lo beneficien antes de que comience la ejecución de la blockchain para luego tener la cadena válida más larga y que esta sea aceptada.

La forma de protegerse de esto es utilizar algo impredecible. Además, para cualquier cadena C, ϵ es un prefijo de C ($\epsilon \preceq C$), por ende, es un valor que funciona bien como "caso base".

3. Refer to Algorithm 1 (validate) in [GKL15], which implements the *chain validation predicate*.

   (a) (5 points) Rewrite validate so that it starts checking from the *beginning* of the chain.

Solution:

```
1:  function validateInverted(C)
2:      b ← V(x_C)
3:      if b ∧ (C ≠ ε) then    ▷ Dejo igual la verificacion de que no sea trivial
4:          i ← 0
5:          repeat
6:              ⟨s, x, ctr⟩ ← C[i]
7:              s' ← C[i + 1].first
8:              if validblock_q^T(⟨s, x, ctr⟩) ∧ (H(ctr, G(s, x)) = s') then
9:                  i ← i + 1
10:             else
11:                 b = False
12:             end if
13:         until i = len(C) − 1 ∨ b = False
14:         b ← validblock_q^T(⟨s, x, ctr⟩)      ▷ No verifico s en el ultimo bloque
15:     end if
16:     return b
17: end function
```

La idea del algoritmo es indexar C para poder ver los bloques en orden, guardando en cada paso en s' el valor de s del bloque siguiente para usar en la función de validación. Al final realiza las mismas comparaciones que el algoritmo mostrado en clase.

Nota: Debería cambiarse el Repeat/Until por un While para que no falle la ejecución al intentar acceder al segundo bloque si se quiere verificar una cadena de un solo bloque, pero no encontré la forma de que LaTeX me tome el bloque While.

   (b) (5 points) Discuss pros and cons of this approach, compared to Algorithm 1's.

Solution: La mayor desventaja de este algoritmo es que, en caso de que el último bloque (o uno de los últimos) sea inválido, tardaría mucho más en detectarse dado que primero revalida toda la cadena.

Esta desventaja para ciertas situaciones puede ser una ventaja, ya que, en caso de querer buscar la subcadena más larga válida de entre varias candidatas, se podría adaptar fácilmente el algoritmo para que la extraiga. Si se quiere buscar esto con el otro algoritmo, sería más difícil adaptarlo y podrían evalu-

arse varios bloques potencialmente inválidos antes de comenzar a computar sólo los bloques válidos de la subcadena válida.

4. **Smart contract programming:** *Matching Pennies*. This assignment will focus on writing your own smart contract to implement the Matching Pennies game. The contract should allow two players (A, B) to play a game of Matching Pennies at any point in time. Each player picks a value of two options—for example, the options might be {0, 1}, {'a', 'b'}, {True, False}, etc. If both players pick the same value, the first player wins; if players pick different values, the second player wins. The winner gets 0.1 ETH as reward. After a game ends, two different players should be able to use your contract to play a new game.

   Example: Let A, B be two players who play the game, each with 0.5 ETH. A picks 0 and B picks 0, so A wins. After the game ends, A's balance is 0.6 ETH (perhaps minus some gas fees, if necessary).

   You should implement the smart contract and deploy it on the course's Sepolia Testnet. Your contract should be as *secure*, *gas efficient*, and *fair* as possible. After deploying your contract, you should engage with other student's contract and play a game on his/her contract. Before you engage with a fellow student's smart contract, you should evaluate their code and analyze its features in terms of security and fairness (refer to the 'Security Vulnerabilities' lecture). You should provide:

   (a) (10 points) The code of your contract, together with a detailed description of the high-level decisions you made for the design of your contract, including:

   - Who pays for the reward of the winner?
   - How is the reward sent to the winner?
   - How is it guaranteed that a player cannot cheat?
   - What data type/structure did you use for the pick options and why?

   Solution: Mi primera idea fue guardar la decisión del primer jugador y luego compararla con la del segundo, pero por la naturaleza de la blockchain, el segundo jugador podría ver la decisión del primero en todo momento, por ende, pensé una alternativa para evitar que el segundo jugador pueda conocer la decisión del primero.

   Para saber esto me inspiré en mi poco conocimiento sobre Zero Knowledge Proof y se me ocurrió guardar de alguna forma encriptada la decisión del primer jugador. Para eso, se debe revelar un número antes de que el segundo jugador tome su elección, que no le permita al segundo jugador conocer qué eligió el primero pero no le permita al primer jugador cambiar su decisión a posteriori.

   Para esto se me ocurrió publicar un número que sea el resultado de un hash de la elección del jugador con un nonce, para el cual el primer jugador debería mostrar la preimagen luego de que el segundo jugador haya tomado su decisión.

Me parecía que esta solución era vulnerable a dos situaciones: 1 - Que un jugador que logre una colisión de la función de hash en algún valor cualquiera, y luego utilice esos valores conociendo la elección del segundo jugador 2 - Si se vuelve a usar el mismo nonce en algún momento, el segundo jugador puede predecir el valor elegido por el primero

ChatGPT me ayudó con estos detalles y me recomendó agregar un contexto, por ende, ahora el hash será compuesto de 3 valores:

$$HashP1 = H(nonce||value||context)$$

El contexto es determinado al momento de utilizar el contrato (por ejemplo, el timestamp del juego). El nonce es a elección del primer jugador y el 'value' es 0 o 1 (''cara o cruz'').

Este valor no aporta información al segundo jugador sobre la elección del primero, pero asegura que el primero no pueda cambiar su elección una vez que fue tomada.

Puede verse el código completo del contrato aquí.

- El contrato se encarga de pagar el premio al ganador
- Se envía con la función ''sent''
- Para garantizar que no se pueda hacer trampa utilicé las técnicas descritas en la idea del algoritmo y agregué una función ''cleanGame()'' que permite reiniciar el juego, en caso de que uno de los jugadores no haya completado su jugada.
- Los datos están guardados en variables globales. Limité el contrato a una sola instancia de juego posible por vez para simplificar el proceso y que sea más sencillo ver posibles fallas de seguridad. Para permitir multiples juegos se podrían usar diccionarios y/o listas (similar a ''bank'').

(b) (5 points) A detailed gas consumption evaluation of your implementation, including:

- The cost of deploying and interacting with your contract.
- Whether your contract is *fair* to both players, including whether one player has to pay more gas than the other and why.
- Techniques to make your contract more cost efficient and/or fair.

Solution: El contrato gastó aproximadamente 90.000 de gas para crearse, y gasta aproximadamente 80.000 para recibir transacciones. Luego, el costo del contrato lamentablemente es asimétrico, dado que, aunque ambos jugadores deben depositar una vez, el primer jugador debe calcular su hash, enviarlo a la blockchain y luego enviar la verificación mientras que el segundo jugador solo debe mostrar su decisión. Luego, el costo podría emparejarse más si el segundo jugador llama a la función "newGame". De este modo, ambos jugadores deberían depositar una vez y luego llamar a dos funciones cada uno.

El costo de las funciones player1Play y player2Play son casi idénticos, poco más de 51.150 de gas. Sin embargo, los costos de las funciones newGame y player1Reveal es más difícil de calcular, la IDE de Remix no logra predecir el coste.

Por último, el primer jugador podría revisar en la blockchain que valor puso el 2do jugador y, en caso de saber que perdió, no llamar a la función de player1Reveal. Aunque hacer esto no le hace recuperar su inversión ni evitar que el jugador 2 reciba el pago (dado que si pasó un día sin revelar la función "cleanGame" le pagará al segundo jugador por defecto), sí que le ahorrará el coste de gas.

Por ende, es complicado hacer que el juego sea simétrico y justo en cuanto al coste de las transacciones.

(c) (5 points) A thorough list of potential hazards and vulnerabilities that *may* occur in your contract; provide a detailed analysis of the security mechanisms you use to mitigate such hazards.

Solution:

- Para evitar publicar la decisión del primer jugador utilicé las técnicas descritas anteriormente
- Para evitar que el segundo jugador elija valores inválidos limité su elección a un valor booleano. Notar que estoy permitiendo al primer jugador elegir un valor distinto a 0 o 1, pero si lo hiciera, se aseguraría perder, ya que el primer jugador gana si ambas monedas son iguales.
- Para evitar que se intente retirar dos veces el premio se modifican "p1Paid" y "p2Pay" a false antes de transferir el premio.
- Para evitar que terceros puedan usar el contrato, verifico que los mensajes sean enviados por uno de los jugadores

- Mi contrato es vulnerable a DOS, si alguien utiliza dos de sus cuentas para jugar y nunca termina el juego. Sin embargo, hacer esto no tendría ninguna utilidad para el atacante y siempre se puede desplegar un nuevo contrato para hacer más juegos.
- Otra vulnerabilidad que podría llegar a tener mi contrato es el uso de ”push” para otorgar las ganancias, sin embargo, no se me ocurrió una forma de aprovechar esto para vulnerar mi contrato, y me pareció mucho más simple hacerlo de esta manera.

(d) (5 points) A description of your analysis of your fellow student's contract (along with relative code snippets of their contract, where needed for readability), including:

- Any vulnerabilities discovered?
- How could a player exploit these vulnerabilities to win the game?

Solution: El principal problema que veo del contrato de mi compañero es que no tiene una función para recuperar lo apostado en caso de que el otro jugador no colabore confirmando su elección en caso de darse cuenta de que perdió. Puede darse cuenta de que perdió fácilmente viendo el nonce del prime jugador luego de que este confirme.

```
function enterGame(uint8 choice, uint256 nonce) payable public returns (uint256){    infinite gas
    require(player1 == address(0) || player2 == address(0));
    require(msg.value > 0);
    require(choice == 0 || choice == 1);
    require(!isAPlayer(msg.sender));

    if(player1 == address(0)){
        player1 = msg.sender;
        confirmed[player1] = false;
    } else if(player2 == address(0)) {
        player2 = msg.sender;
        confirmed[player2] = false;
    }
    hashed_choices[msg.sender] = keccak256(abi.encodePacked(nonce, choice));
    return nonce;
}
```

```
function confirm(uint256 nonce) public {    infinite gas
    require(player1 != address(0) && player2 != address(0));
    require(isAPlayer(msg.sender));

    if(!confirmed[msg.sender]){
        confirmed[msg.sender] = true;
        getChoice(nonce);
        if(confirmed[player1] && confirmed[player2]){
            play();
        }
    }
}
```

Luego, como solo se utiliza un nonce elegido por el usuario y el valor binario, en caso de que se encuentre una colisión en la función de hash sería altamente probable que un usuario pueda elegir dos nonces que produzcan el mismo hash para valores distintos. Luego, al momento de confirmar, puede esperar a que confirme el otro jugador y luego usar el nonce que se decodificará como la elección que le de la victoria.

```
function getChoice(uint256 nonce) private {    infinite gas
    if(hashed_choices[msg.sender] == keccak256(abi.encodePacked(nonce, uint8(1)))){
        choices[msg.sender] = 1;
    } else if (hashed_choices[msg.sender] == keccak256(abi.encodePacked(nonce, uint8(0)))) {
        choices[msg.sender] = 0;
    } else {
        revert("Invalid nonce.");
    }
}
```

(e) (5 points) The transaction history of an execution of a game on your contract.

Solution:

CreateGame

Player1 Transact (payment)

Player1Play

Player2 Transact (payment)

Player2Play

Player1Reveal