Fundamentos y Aplicaciones de Blockchains

Homework 2

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

Student: Felipe Luc Pasquet *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: Nosotros vimos PKI como una configuración confiable (*trusted setup*) para lograr consenso. En la Infraestructura de Clave Pública (*PKI*), los mensajes son firmados con firmas digitales por entidades confiables, que pueden ser comprobadas fácilmente por todos los usuarios (porque todos conocen la clave pública de las entidades confiables) pero no pueden ser creadas sin la clave privada del autor de la fima.
   Las firmas digitales nos garantizan integridad de los mensajes, es decir, no pueden ser modificados.
   El protocolo de consenso de Bitcoin es distinto al de PKI, pero también utiliza firmas digitales.
   Una de las diferencias es que el protocolo de consenso de bitcoin es sincronizado parcialmente únicamente, y los mensajes no se comparten punto a punto, si no que se van difundiendo a toda la red de a poco. Además es una configuración pública, en la que cualquiera puede participar, y no hay "entidades autenticadas".
   Sin embargo también se utilizan las firmas digitales para firmar todos los mensajes de la red, por lo que todos los mensajes mantienen las propiedades de seguridad de las firmas digitales.
   Además en el protocolo de bitcoin se necesita que $n > 2t$ para que halla consenso, lo cual es una cota bastante buena

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

   Solution: No, el protocolo de Bitcoin no asume PKI. Justamente asume que la configuración es pública, a diferencia de PKI, y que cualquiera usuario puede participar y firmar mensajes en la red. El protocolo de bitcoin asume que los usuarios honestos van a aceptar únicamente la cadena más larga en la que todos los bloques son validos. Este protocolo funciona mientras $n > 2t$.

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: En clase vimos como se usaba un string impredecible (la primer plana del diario de ese día) en el bloque genesis. Sin embargo esto no es algo estrictamente necesario para que funcione el algoritmo. El algoritmo funciona incluso si el primer bloque es un string vacío, pero el problema es que si todos saben que una blockchain va a comenzar con un primer bloque vacío (o con cualquier string, pero ya lo conocen) podrían crear una cadena de bloques valida antes de que empiece la blockchain oficialmente y luego difundirla a los demás participantes. Si se conoce cual va a ser el bloque inicial, podrían crear toda una cadena de bloques validos antes que los demás participantes/mineros lo sepan y luego el resto de participantes tendrían que aceptarla porque sería la cadena más larga.

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.

Algorithm 1 Validate(C)

$b \leftarrow True$
$n \leftarrow len(C)$
if $n \geq 1$ **then**
  $\langle s, x, ctr \rangle \leftarrow C^{|(n-1)}$
  $b \leftarrow validblock_q^T(\langle s, x, ctr \rangle)$
  $s' \leftarrow H(ctr, G(s, x))$
  $n \leftarrow n - 2$
  while $b = True$ and $n \geq 0$ **do**
    $\langle s, x, ctr \rangle \leftarrow head(C^{|n})$
    if $s' \neq s$ or $\neg validblock_q^T(\langle s, x, ctr \rangle)$ **then**
      $b \leftarrow False$
    end if
    $s' \leftarrow H(ctr, G(s, x))$
    $n \leftarrow n - 1$
  end while
end if
Return b

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

Solution: Uno de los pros de validar bloques desde el principio de la cadena es que todos los bloques que se verifican hasta el momento sabemos con total seguridad que son validos, mientras que en el otro algoritmo no podemos estar seguros de ningún que bloque sea valido hasta que terminamos de ejecutar el código, que es el primer bloque de la cadena.
Por el otro lado, de con este algoritmo llegamos al nodo que más nos importa verificar generalmente (el último en ser agregado) al final del código, por lo que si lo hacemos de esta manera vamos a tener que esperar a que termine de verificar toda la cadena para saber si el último bloque agregado es valido, mientras que el algoritmo visto en clase nos da una respuesta parcial más rápidamente.

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?

```
pragma solidity ^0.8.0;

contract MatchingPennies {
    address public playerA;
    address public playerB;
    uint public betAmount;

    enum Option {None, Heads, Tails}
    Option private playerAChoice;
    Option private playerBChoice;

    enum GameState {WaitingForPlayers, WaitingForChoices, GameEnded}
    GameState public gameState;

    mapping(address => uint) public balances;

    // Events for logging actions
```

```
event GameStarted(address playerA, address playerB, uint betAmoun
event PlayerMadeChoice(address player, Option choice);
event GameEnded(address winner, uint reward);

modifier onlyPlayers() {
    require(msg.sender == playerA || msg.sender == playerB, "Only
    _;
}

modifier inState(GameState state) {
    require(gameState == state, "Invalid game state for this acti
    _;
}

// Constructor to initialize the game
constructor(address _playerB, uint _betAmount) {
    playerA = msg.sender;
    playerB = _playerB;
    betAmount = _betAmount;
    gameState = GameState.WaitingForChoices;
    balances[playerA] = 0;
    balances[playerB] = 0;
    emit GameStarted(playerA, playerB, betAmount);
}

// Function for Player A to make a choice
function makeChoiceA(Option choice) external payable onlyPlayers
    require(msg.sender == playerA, "Only Player A can make this c
    require(msg.value + balances[playerA]>= betAmount, "Insuffici
    balances[playerA] += msg.value;
    playerAChoice = choice;
    checkGameEnd();
}

// Function for Player B to make a choice
function makeChoiceB(Option choice) external payable onlyPlayers
    require(msg.sender == playerB, "Only Player B can make this c
    require(msg.value + balances[playerB] >= betAmount, "Insuffic
    balances[playerB] += msg.value;
    playerBChoice = choice;
    checkGameEnd();
}
```

```
    // Check if both players have made a choice and determine the win
    function checkGameEnd() internal {
        if (playerAChoice != Option.None && playerBChoice != Option.N
            //address winner = determineWinner();
            gameState = GameState.GameEnded;
            if (playerAChoice == playerBChoice) {
                balances[playerA] -= betAmount;
                balances[playerB] += (betAmount);  // Player B wins i
            } else {
                balances[playerB] -= betAmount;
                balances[playerA]+= (betAmount);  // Player A wins if
            }
        }
    }


    // Reset the game and allow new game
    function resetGame(address _newPlayerB, uint _betAmount) external
        require(msg.sender == playerA, "Only Player A can reset the g
        playerB = _newPlayerB;
        gameState = GameState.WaitingForChoices;
        playerAChoice = Option.None;
        playerBChoice = Option.None;
        betAmount = _betAmount;
        emit GameStarted(playerA, playerB, betAmount);
    }

    // View function to get the balance of a player
    function getBalance() external view returns (uint) {
        return balances[msg.sender];
    }

    function withdraw() external payable {
        require(gameState != GameState.WaitingForChoices, "You cannot
        require(balances[msg.sender]>0, "No money on balance");
        uint amount = balances[msg.sender];
        balances[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }
}
```

Solution: en el siguiente código programé el juego como se pedía. El creador del contrato especifica cuanto se va a apostar y con quien va a jugar. Luego de que ambos jugadores hicieron su apuesta, se suma la ganancia al balance del ganador (y se resta del balance del perdedor). Solo se puede retirar el balance cuando la partida ya finalizó, para evitar que retiren sus fondos luego de hacer una apuesta.

(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 costo de crear el contrato fue 0.00217124
El costo de hacer una apuesta para el jugador A fue el valor de la apuesta y 0.00008241 de gas en las partidas
y el costo de hacer una apuesta para el jugador B fue el valor de la apuesta + 0.00008899 de gas en la primer partida y 0.00007694 en la segunda partida)
El costo de retirar las ganancias fue para ambos jugadores 0.00004643 en gas.
Concluimos que es bastante justo en el reparto del gas entre ambos jugadores.

(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: Actualmente, como el contrato guarda en la blockchain la información de que decisión tomo cada jugador, un jugador podría esperar a que el otro jugador tome una decisión para ver que decidio, y ganar siempre. No encontré cual sería la solución para esto.
También podría pasar que un jugador nunca tome una decisión, y el juego no se podría resetear ni nadie podría retirar su dinero.

(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: Analicé el código de **Gaspar Zucker**, y la verdad que no encontré ninguna vulnerabilidad. Resolvió de manera muy inteligente las dos vulnerabilidades que tenía mi código. Por un lado, los jugadores juegan por turnos. El primer jugador manda su elección, pero luego de pasar por una función hash junto con un nonce de su elección. De esta manera, lo que manda es un hash que el segundo jugador no va a poder descifrar que es. Luego el segundo jugador tiene que hacer su apuesta. Finalmente el primer jugador tiene que revelar cual era el nonce y su suposición original, antes de las 24hs, y se procede a ver quien es el ganador y transferirle el premio correspondiente.
Si el jugador 1 no revela su supocición antes de las 24hs gana automáticamente el jugador 2, por lo que no puede trabar el programa ni los fondos del contrato como sucedía en mi contrato.
Lo único que puedo ver como ”falla”, o injusto de este contrato es que el jugador 1 va a tener que pagar más gas que el jugador 2, ya que tiene que ejecutar más funciones. Se me ocurre que esto se podría resolver cobrandole al jugador 2 una comisión para igualar los costos y que no haya preferencia de un jugador sobre el otro.

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

Solution:
DirecciÓn del contrato:
0xD75C8B913e2B658115EE653A4f507ebEE0Ee9375

Player A choice:
0x41991fd9858a535f9fd339ce1a6142fccce24fe24ffeb4942b14ecc67f71c9ce

Player B choice:
0x2b6ed8e7e5cfe291eb7a67be07f3fe8d62a67eb960fa1a61e82b4e5a562f440a

Player B withdraw:
0xad7fb9b183778ea3063041239e3b96a1f7a1ad60c7637726719b4453f8acf2f3

Transfer to Player B:
0xad7fb9b183778ea3063041239e3b96a1f7a1ad60c7637726719b4453f8acf2f3