Fundamentos y Aplicaciones de Blockchains, Homework 2

Matías Eliel Waisman

Ejercicio 1

Punto A

El propósito de una Public Key Infrastructure es incrementar la confianza entre los participantes de una red. Permite verificar y confiar en las firmas digitales emitidas por la entidad certificadora, garantizando la autenticidad de los mensajes y la identidad de los participantes.

La PKI lo que nos garantiza es que dada una clave publica la podes asociar a una entidad/ persona. Si el problema de concenso requiere poder asociar a una entidad con una clave, entonces vamos a necesitar tener una PKI.

Un ejemplo en el cual nos beneficia tener una PKI es para para poder resolver los problemas de consenso bizantino con una mayor cantidad de participantes maliciosos. En la siguiente tabla comparo la cantidad de participantes maliciosos máximos que podemos tener para llegar al consenso si no tenemos setup vs si lo tenemos:

Problema	Máximo número de participantes maliciosos sin Setup	Máximo número de participantes m
Consenso	$\frac{N}{3} - 1$	$\frac{N}{2} - 1$
Broadcast	$\frac{N}{3} - 1$	$N - 1$

Pero si nuestro problema de concenso se puede resolver sin necesidad de asociar una firma a una entidad, la PKI no es la que garantiza las propiedades de las firmas digitales.

(Hablar mas acerda de las propiedades de las firmas digitales y pq PKI no te garantiza eso)

Punto B

En la red de Bitcoin, a diferencia de un sistema distribuido tradicional, los nodos no tienen identidades fijas ni están registrados. La red es dinámica, porque cualquiera puede entrar o salir libremente y no hay un listado de participantes.

Por este motivo, el protocolo de consenso de la blockchain corre sin depender de una PKI. En particular, no necesitamos asociar una clave a una entidad para autenticar a los mineros, sino que se usa Proof Of Work para autenticar y generar consenso en el problema de quién va a ser el que agregue el bloque a la blockchain.

Sin embargo, las transacciones sí emplean firmas digitales mediante ECDSA. Cuando un usuario crea una transacción, la firma con su clave privada, y los demás participantes

pueden verificar la validez de esa firma usando la transacción y la clave pública del usuario que firmó, para verificar que él fue el que hizo la transacción.

La diferencia clave con un esquema de PKI es que en Bitcoin no existe una autoridad central que emita o certifique las claves que se usan para firmar transacciones. Cada usuario genera su propio par de claves para firmar transacciones localmente.

Esto ocurre porque en la red de Bitcoin no nos interesa relacionar una clave publica con un usuario/ ente. Por lo tanto en Bitcoin no se usa PKI.

Nota: preguntar qué onda lo del string público del principio porque suena a PKI.

Ejercicio 2

El algoritmo arranca del bloque epsilon, y no de un genesis aleatorio, porque asume que no van a haber ataques de pre computación.

(Hablar de problema de pre computacion y como te salva el bloque genesis aleatorio) (Fuente)

Ejercicio 3

Punto A

Mi idea es usar los prefijos de la cadena y quedarme siempre con la cabeza de ese prefijo para verificar la validez de los bloques del más viejo al más reciente.

Algorithm 1 Validación de la cadena desde el bloque más viejo

```
1: function VALIDATE(C)
2:     b ← V(x_C)
3:     if b ∧ (C ≠ ε) then
4:         ⟨s, x, ctr⟩ ← head(C^[1])                ▷ Me quedo con el bloque más viejo
5:         if validblock^q_T(⟨s, x, ctr⟩) then        ▷ Si el primer bloque es válido
6:             i ← 2
7:             repeat▷ Repito hasta que i sea mayor a la longitud de la cadena o b sea falso
8:                 ⟨s', x, ctr⟩ ← head(C^[i])        ▷ Siguiente bloque al más viejo
9:                 if validblock^q_T(⟨s', x, ctr⟩) ∧ (H(ctr, G(s', x)) = s) then ▷ Valido bloque y
   referencia
10:                    s ← s'                        ▷ Guardo el hash del bloque actual
11:                else
12:                    b ← False
13:                end if
14:                i ← i + 1
15:            until (i = len(C) + 1) ∨ (b = False)
16:        end if
17:    end if
18:    return b
19: end function
```

Punto B

Comenzar la validación desde el primer bloque (el más antiguo) garantiza que cada bloque que consideramos válido realmente lo es, ya que se verifica la validez de toda la cadena de manera constructiva. Esto evita confiar en bloques mas viejos para validar bloques mas nuevos, lo cual podría ocultar errores si un bloque apunta a otro inválido. En cambio al verificar del mas viejo al mas nuevo, nos aseguramos que todos los bloques que afirmamos que son validos realmente lo son. Te da un invariante el algoritmo.

Una desventaja de comenzar la validación desde el primer bloque es que, en la práctica, los errores de validez son mucho más probables en los bloques más recientes. Por ejemplo, validar el bloque génesis no aporta demasiado, ya que siempre se asume correcto y es altamente improbable que allí esté el error. En consecuencia, este enfoque podría implicar un costo innecesario de verificación de muchos bloques 'viejos" antes de detectar una invalidez en un bloque más reciente. En términos de complejidad asintótica, ambos algoritmos tienen la misma cota superior (en el peor caso deben verificar toda la cadena). Sin embargo, en la práctica resulta más eficiente comenzar desde el bloque más reciente hacia atrás, ya que aumenta la probabilidad de encontrar un bloque inválido más temprano y, por lo tanto, reducir la cantidad de bloques que es necesario verificar.

Ejercicio 4

Punto A

El que pierde va a ser el que pague la recompensa al que gano.

Para seguir el patron de pull over push, el ganador va a tener que llamar a withdraw, en vez de hacerle un send o transfer.

Si se logea el evento y se puede enterar todo el mundo para evitar que el jugador 2 siempre espere a que juegue el 1 y luego lea la blockchain para ver el valor que eligio. El usuario va a mandar el mensaje junto a un string aleatorio, siempre de la misma longitud, y despues el contrato va a desencriptarlo con la clave publica.

El primer código que hicimos con Segundo Sacchi tiene el problema de que si el jugador 2 ve lo que manda el jugador 1 al contrato, va a ganar siempre.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Pennies{
    bool jugo_primero = false; // Cuando juegue el primer
       jugador lo pongo en True, cuando juegue el segundo lo
       pongo en False
    address addr_primero;
    address addr_segundo;
    bool valor_primero;
    mapping(address => uint256) balance;
    address[] public customers;
```

```
        event Withdrawal(address customer);

        function withdraw() public {
            uint256 b = balance[msg.sender];
            balance[msg.sender] = 0;
            payable(msg.sender).transfer(b);
            emit Withdrawal(msg.sender);
        }

        function jugar(bool valor) public payable{
            require(msg.value >= 0.01 ether); // Si manda mas se lo
               queda la casa como donacion
            if(jugo_primero){
                // Tenemos a los dos jugadores
                jugo_primero = false;
                addr_segundo = msg.sender;
                if(valor_primero == valor){
                    balance[addr_primero] += 0.02 ether;
                }
                else{
                    balance[addr_segundo] += 0.02 ether;
                }
            }
            else{
                // Estamos esperando al segundo
                addr_primero = msg.sender;
                jugo_primero = true;
                valor_primero = valor;
            }
        }
    }
```

Por lo que