Raspodijeljene glavne knjige i kriptovalute

Završni ispit – 6. veljače 2019.

1. **(8 bodova)** Razmatramo Ethereum sustav:

   a) **(2 boda)** Tko sve izvršava kôd pametnog ugovora, tko plaća troškove izvođenja, a tko prima nagradu za rudarenje?

   b) **(2 boda)** Objasnite ulogu *gasa* i kako se definira nagrada za izvršavanje koda pametnog ugovora.

   c) **(2 boda)** Što je to *ommer* (uncle) blok i koja je motivacija za njegovo uvođenje u Ethereumu?

   d) **(2 boda)** Kada *ommer* blok prestaje biti ispravan i što se događa s transakcijama u njemu?

2. **(8 bodova)** Budući da s jedne strane radite i primate plaću preko Studentskog centra, a s druge strane svakodnevno koristite usluge studentske menze, razmatrate korištenje Lightning Networka na Bitcoin blockchainu kako biste ostvarili kanal plaćanja između vas i Studentskog centra.

   a) **(3 boda)** Ukratko objasnite kako funkcionira Lightning Network i zašto bi transakcije preko Lightning Networka bile brže od onih na blockchainu.

   b) **(3 boda)** Ukoliko otvorite kanal, primite jednu plaću, 5 puta platite obrok u menzi, te nakon toga odlučite zatvoriti kanal, koliko ukupno blockchain transakcija nastane u ovom slučaju?

   c) **(2 boda)** Može li Lightning Network u potpunosti zamijeniti blockchain transakcije? Obrazložite odgovor.

3. **(8 bodova)** Pretpostavimo da u sustavu određene kriptovalute postoje dvije nezavisne grupe rudara A i B koje implementiraju različite verzije protokola. U jednom trenutku neki napadač pronađe ranjivost u implementaciji A zbog koje će rudari prihvaćati transakcije koje dvostruko troše neki UTXO. Rudari koji koriste verziju B takve transakcije neće smatrati valjanima.

   a) **(4 boda)** Ukoliko je 80% rudarske snage na verziji A (koja sadrži ranjivost), a 20% na (ispravnoj) verziji B, što će se dogoditi s lancem blokova kad rudar s verzijom A predloži blok u kojem postoji neispravna transakcija?

   b) **(4 boda)** Što će se dogoditi u obrnutom slučaju (dakle 80% rudarske snage je na verziji B, a 20% na verziji A)?

4. **(8 bodova)** Razmišljate o investiciji u kriptovalute i razmatrate na koje načine je možete ostvariti.

   a) **(4 boda)** Ako novčiće kupite na centraliziranoj burzi kriptovaluta, u kojem trenutku se transakcija između vas i prodavača zapisuje u blockchain te kriptovalute? Kad se to događa u decentraliziranim burzama?

   b) **(3 boda)** Navedite i opišite bar jedan način na koji je moguće osigurati stabilnost cijene stablecoina.

   c) **(1 bod)** Kupili ste 1 Bitcoin u 2015. godini (kad je bio znatno jeftiniji), te ste ga sad (u 2019. godini) odlučili prodati. Morate li platiti porez, i ako da, na koji iznos i po kojoj stopi?

5. **(8 bodova)** Na drugoj stranici ispita nalazi se Solidity kôd pametnog ugovora koji omogućuje korisnicima kupnju i prodaju karata za događaje koristeći Ethereum platformu. Svi korisnici imaju pravo napraviti ponudu karata za neki događaj (pozivanjem metode `createEvent`), te bilo tko ima pravo kupiti karte iz ponude (metoda `buyNew`), dok god ima neprodanih karata. Nakon što korisnik kupi kartu on je može ponuditi po novoj cijeni (`offer`), te drugi korisnik može prihvatiti tu ponudu kako bi kupio kartu od njega (`buyOffered`).

   Nažalost ovaj ugovor je napisala osoba koja nije dobro pročitala dokumentaciju Solidity jezika te nije položila RGKK na FER-u, zbog čega ugovor sadrži nekoliko grešaka. Vaš zadatak je pronaći barem 4 greške, opisati koje su posljedice tih grešaka te predložiti kako se taj dio koda može popraviti. (Pronalazak jedne greške, opis posljedica i ispravak nosi 2 boda)

```
pragma solidity ^0.4.0;

contract TicketDepot {

  struct Event{
    address owner;
    uint64 ticketPrice;
    uint16 ticketsRemaining;
    mapping(uint16 => address) attendees;
  }

  struct Offering{
    address buyer;
    uint64 price;
    uint256 deadline;
  }

  uint8 numEvents;
  address owner;
  uint64 transactionFee;
  mapping(uint8 => Event) events;
  mapping(bytes32 => Offering) offerings;

  function ticketDepot(uint64 _transactionFee) public {
      transactionFee = _transactionFee;
      owner = tx.origin;
    }

  function createEvent(uint64 _ticketPrice, uint16 _ticketsAvailable) returns (uint8) {
    numEvents++;
    events[numEvents].owner = tx.origin;
    events[numEvents].ticketPrice = _ticketPrice
    events[numEvents].ticketsRemaining = _ticketsAvailable;
      return numEvents; // This is eventID
  }

  modifier available(uint8 _eventID) {
    _;
    if (events[_eventID].ticketsRemaining <= 0) throw;
  }

  function buyNew(uint8 _eventID, address _attendee) available(_eventID) payable
     returns (uint16) {
    if (msg.sender == events[_eventID].owner || msg.value > events[_eventID].
       ticketPrice + transactionFee){
      ticketID = events[_eventID].ticketsRemaining--;
      events[_eventID].attendees[ticketID] = _attendee;
      events[_eventID].owner.send(msg.value - transactionFee);
      return ticketID;
    }
  }

  function offer(uint8 _eventID, uint16 _ticketID, uint64 _price, address _buyer,
     uint16 _offerWindow) {
    if (msg.value < transactionFee) throw;
    bytes32 offerID = sha3(_eventID + _ticketID);
    if (offerings[offerID] != 0) throw;
    offerings[offerID].buyer = _buyer;
    offerings[offerID].price = _price;
    offerings[offerID].deadline = block.number + _offerWindow;
  }

  function buyOffered(uint8 _eventID, uint16 _ticketID, address _newAttendee) payable {
    bytes32 offerID = sha3(_eventID + _ticketID);
    if (msg.value > offerings[offerID].price && block.number < offerings[offerID].
       deadline &&
       (msg.sender == offerings[offerID].buyer || offerings[offerID].buyer == 0) {
        events[_eventID].attendees[_ticketID].send(offerings[offerID].price);
        events[_eventID].attendees[_ticketID] = _newAttendee;
        delete offerings[offerID];
      }
  }
}
```