Raspodijeljene glavne knjige i kriptovalute
Zimski ispitni rok – 20. veljače 2019.

1. **(20 bodova)** Razmatramo Bitcoin protokol konsenzusa i strukturu blokova.

a) **(6 bodova)** Kakvu kriptografsku slagalicu rudari u Bitcoin sustavu moraju riješiti kako bi predložili novi blok?

b) **(6 bodova)** Zašto bi rudari htjeli uključiti što više transakcija u blok koji predlažu i čime je taj broj ograničen?

c) **(6 bodova)** Objasnite što je to Merkleovo stablo i skicirajte kako bi izgledalo Merkleovo stablo koje koristi hash funkciju $H()$ i sadrži 4 transakcije: $tx1$, $tx2$, $tx3$ i $tx4$.

d) **(2 boda)** Zašto zaglavlje bloka sadrži korijen Merkleovog stabla transakcija?

2. **(20 bodova)** Želite kupiti stan i uštedili ste dovoljno Bitcoina za to, ali s druge strane imate svakodnevne životne troškove za koje su vam također potrebni Bitcoini, pa razmatrate načine na koje možete pohraniti svoje novčiće.

a) **(6 bodova)** Što je topla pohrana (*hot storage*), a što hladna pohrana (*cold storage*)? Objasnite njihove glavne prednosti i mane.

b) **(2 boda)** Koju od ovih metoda biste odabrali za čuvanje novaca za stan, a koju za svakodnevne troškove?

c) **(6 bodova)** Što je to hijerarhijski novčanik? Koja je razlika u odnosu na klasično generiranje adresa (preko funkcije `generateKeys`)?

d) **(6 bodova)** Što je to tajno dijeljenje i kako ono rješava problem *single point of failure*? Objasnite na primjeru kada je ključ podijeljen na 5 dijelova, a potrebno je znati 2 od tih 5 za rekonstrukciju ključa.

3. **(20 bodova)** Razmatramo anonimnost i pitanje regulative kriptovaluta. Kažemo da je korisnik anoniman ako se njegove različite interakcije sa sustavom ne mogu povezati. Jedno rješenje za povećanje nepozivosti je tehnika koju zovemo miješanje (engl. *mixing*).

a) **(8 bodova)** Zašto možemo reći da su online novčanici primjer miješanja i koji su njegovi glavni nedostaci?

b) **(9 bodova)** Združeni novčić (engl. *coinjoin*) primjer je raspodijeljenog miješanja. Objasnite prednosti raspodijeljenog miješanja i protokol *coinjoin*-a.

c) **(3 boda)** Koji je glavni pozitivni argument za uvođenje regulative za kriptovalute? Možete objasniti na primjeru.

4. **(20 bodova)** Pomoću Bitcoin naredbe `OP_CHECKMULTISIG` moguće je implementirati sustav prijenosa Bitcoin novčića koji osigurava obje strane u slučaju sukoba. Npr. Ante želi prodati svoj proizvod Zvonku za 1 Bitcoin. U slučaju da Zvonko nakon primanja proizvoda ne želi platiti Anti, Ante može dobiti svoje novčiće tako da neki pošteni sudac uz Antu potpiše transakciju. U slučaju da je Ante poslao proizvod koji nije ono što je Zvonko očekivao, ni Zvonko ni sudac neće htjeti potpisati transakciju pa Zvonko neće izgubiti svoje novčiće.

a) **(8 bodova)** Napišite `pubScript` (*locking* skriptu) koja omogućava takav sustav.

b) **(12 bodova)** Napišite `pubScript` (*locking* skriptu) za sustav u kojem postoje 3 sudca od kojih bilo tko smije potpisati transakciju u slučaju sukoba između Zvonka i Ante. Skripta mora osigurati da sudci ne mogu potpisati transakciju bez barem jednog potpisa Zvonka ili Ante.
Napomene: Svaki sudionik u transakciji može imati više od jednog javnog i privatnog ključa. Također, u bodovanje ulazi i veličina skripte s obzirom na broj operacija i veličinu u byteovima.

5. **(20 bodova)** Na drugoj stranici ispita nalazi se Solidity kôd pametnog ugovora koji omogućuje korisnicima igranje igre križić-kružić. Ugovor također omogućuje korisnicima klađenje na ishod partije, tj. svaki igrač mora uplatiti neki ulog na ugovor. Svaka instanca ugovora postavljenog na Blockchain predstavlja jednu partiju između dva igrača. Za funkciju `checkGameOver()` možete pretpostaviti da je ispravno implementirana.

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 sigurnosnih propusta. Vaš zadatak je pronaći barem 4 takva propusta koji omogućuju napadaču manipulaciju tijeka igre i/ili povlačenje novaca s ugovora bez obzira na ishod partije. Uz pronalazak propusta potrebno je opisati koje su posljedice tih propusta te predložiti kako se kôd može popraviti da više ne sadrži propust.

Ignorirajte sintaksne greške. Pronalazak jednog propusta, opis posljedica i ispravak nosi 5 bodova. Dopušteno je/preporuča se pronalazak više od 4 propusta. Rješenja koja ne ukazuju na sigurnosne propuste i/ili još više komprimiraju sigurnost ugovora mogu biti nagrađena negativnim bodovima.

```
pragma solidity ^0.4.24;

contract TicTacToe {

    uint32 _turnLength;
    address[2] _players;

    uint8 _p2Nonce;
    bytes32 _p1Commitment;

    uint8[9] _board;
    uint8 _currentPlayer;
    uint _turnDeadline;

    constructor(address opponent, uint32 turnLength, bytes32 p1Commitment) public {
        _players[0] = msg.sender;
        _players[1] = opponent;
        _turnLength = turnLength;
        _p1Commitment = p1Commitment;
    }

    function joinGame(uint8 p2Nonce) public payable {
        require(msg.sender == _players[1]);              // Check player 2.
        require(msg.value >= address(this).balance);     // Check player 2 stake.
        _p2Nonce = p2Nonce;                              // Register player 2 nonce.
    }

    function startGame(uint8 p1Nonce) public {
        require(keccak256(p1Nonce) == _p1Commitment);    // Check player 1 commitment.
        _currentPlayer = (p1Nonce ^ _p2Nonce) & 0x01;    // Select starting player.
        _turnDeadline = block.number + _turnLength;      // Set new turn deadline.
    }

    function playMove(uint8 squareToPlay) public {
        require(msg.sender == _players[_currentPlayer]); // Check that correct player.
        _board[squareToPlay] = _currentPlayer;           // Make a move.
        if (checkGameOver()) {                           // Check if game is finished.
            selfdestruct(msg.sender);                    // Destroy contract and send
        }                                                // founds to winner.
        _currentPlayer ^= 0x01;                          // Set new current player.
        _turnDeadline = block.number + _turnLength;      // Set new turn deadline.
    }

    function endGame() public {
        if (block.number > _turnDeadline) {              // Check deadline.
            selfdestruct(msg.sender);                    // Destroy contract and send
        }                                                // founds to caller.
    }

    function checkGameOver() internal returns (bool gameOver) {
        // Assume correct implementation.
    }
}
```