Final: 4010, 5010 Cosci - Blockchain

Your Name: _____

Short Essay Questions.

1. (30pt) Explain a zero knowledge proof to a layman. Provide an example of how a zero knowledge proof can be used.

2. (30pt) How do blockchains create trust between non-trusting parties?

3. (30pt) What is a blockchain? - Why is blockchain important? - What are blockchains used for?

4. (30pt) What is a smart contract? Provide an Example of what you would use one for.

5. (30pt) What is “mining” in the context of a blockchain system?

6. (30pt) What are the good properties of using a blockchain as a database? Compare to a relational database.

7. (30pt) What are the bad properties of using a blockchain as a database? Compare to a relational database.

8. (30pt) What are the bad properties of blockchain as a database? Compare to a relational database.

9. (30pt) Why are blockchains (Ethereum, Bitcoin) a so slow?

10. (30pt) Explain the Byzantine generals problem and how Threshold-ECDSA solves it.

11. (20pt) Provide an elevator pitch (if you said it out loud you have about 30 seconds) to explain why blockchain is an important technology. Assume that your audience is moderately non-technical.

Coding:

1. Solidity - a simple contract to take the hash of a document. (The code is on the last page of the test)
    1. [20pt] When is the contract constructor run?

    2. [20pt] Who can call the setPayment function on line 19?

    3. [20pt] Explain what the 'newInfo' function performs (line 23 to 31)

    4. [20pt] Explain what "gas" is and how that effects running of contracts

Simple Contract to Register Documents

```
pragma solidity >=0.4.25 <0.9.0;

import "openzeppelin-solidity/contracts/ownership/Ownable.sol";

contract DocumentReg is Ownable {
  using SafeMath for uint256;

  mapping(bytes32 => bool) infoSet;
  mapping(bytes32 => address) infoOwner;
  mapping(bytes32 => string) infoData;
  mapping(bytes32 => string) infoName;
  mapping(address => uint256) infoNDocs;
  mapping(address => []bytes32) infoDocs;

  mapping(address => bool) isNotery;
  mapping(bytes32 => address) noterized;

  uint256 minPayment;
  address contractOwner;

  event DocumentSet(string, indexed bytes32, string);
  event DocumentOwner(indexed address, string, bytes32, string);
  event DocumentNoterized(string, bytes32, string, indexed address);
  event Transfer(address, address, uint256);

  constructor() {
    contractOwner = msg.sender;
    minPayment = 1;
  }

  function setPayment ( uint256 p ) public onlyOwner {
    minPayment = p;
  }

  function setNoterizer ( bytes32 aNotery ) public onlyOwner {
    isNotery[aNotery] = true;
  }

  function rmNoterizer ( bytes32 aNotery ) public onlyOwner {
    isNotery[aNotery] = false;
  }

  function newDocument ( string name, bytes32 infoHash, string info )
  public payable returns(bool) {
    require(!infoSet[infoHash], "already set, already has owner.");
    require(msg.value >= minPayment, "insufficient payment to set data.");
    infoSet[infoHash] = true;
    infoOwner[infoHash] = msg.sender;
    if ( infoNDocs[msg.sender] ) {
      infoNDocs[msg.sender] = infoNDocs[msg.sender] + 1;
    } else {
      infoNDocs[msg.sender] = 1;
    }
    infoDocs[msg.sender].push ( infoHash );
    infoData[infoHash] = info;
    infoName[infoHash] = name;
    emit DocumentSet(name, infoHash, info);
    emit DocumentOwner(msg.sender, name, infoHash, info);
    return true;
  }

```

```
61   function noterizeDocument ( string name, bytes32 infoHash,
     string info ) public returns (bool) {
62     require(infoSet[infoHash], "document not created set.");
63     require(!isNotery[msg.sender], "not a registered notery.");
64     noterized[infoHash] = msg.sender;
65     emit DocumentNoterized(name, infoHash, info, msg.sender);
66     return true;
67   }
68
69   /**
70   * @dev transfer token for a specified address
71   * @param _to The address to transfer to.
72   * @param _value The amount to be transferred.
73   */
74   function transfer(address _to, uint256 _value) public returns (bool) {
75     require(_to != address(0), "Invalid null address.");
76     require(_value <= balances[msg.sender], "Amoutn to transfer is too large.");
77
78     // SafeMath.sub will throw if there is not enough balance.
79     balances[msg.sender] = balances[msg.sender].sub(_value);
80     balances[_to] = balances[_to].add(_value);
81     addToCapTable(_to);
82     emit Transfer(msg.sender, _to, _value);
83     return true;
84   }
85
86   /**
87    * @dev withdraw transfers all of the existing funds from the contract to
        the owner of the contract.
88    */
89   function withdraw() public onlyOwner {
90     contractOwner.transfer(address(this).balance);
91   }
92
93   // List Documents by Owner of Document
94   function nDocuments() public view returns ( uint256 ) {
95     return( infoNDocs[msg.sender] );
96   }
97   function getDocName ( uint256 nth ) public view returns ( string ) {
98     require(nth >= 0  && nth < infoDocs[msg.sender].length, "nth out of range.");
99     bytes32 v = infoDocs[nth];
100    return ( infoName[v] );
101  }
102  function getDocInfoHash ( uint256 nth ) public view returns ( bytes32 ) {
103    require(nth >= 0  && nth < infoDocs[msg.sender].length, "nth out of range.");
104    bytes32 v = infoDocs[nth];
105    return ( v );
106  }
107  function getDocInfo ( uint256 nth ) public view returns ( string ) {
108    require(nth >= 0  && nth < infoDocs[msg.sender].length, "nth out of range.");
109    bytes32 v = infoDocs[nth];
110    return ( infoData[v] );
111  }
112
112 }
```