**CS 426 - Introduction to Blockchains**
**Mid term Exam**
**Duration: 2 Hours**

*Instructions:*

1. *There are 8 questions in this question paper. All questions are compulsory.*
2. *For each question, please justify how you arrived at your answer. Incomplete answers will lead to loss of marks.*
3. *Answer each question using the information provided in the question (no clarifications during the exam).*
4. *Assumptions made in each question should be clearly stated. The assumption should be rational and reasonably accurate. Wild assumptions should not be considered for arriving at an answer.*

**Q1.** Choose the right option(s) (more than one answers are possible, marks will be awarded if all the correct choices corresponding to a given question are selected), [10 Marks]

i. Which of the following bitcoin block parameters is/are not related to Mining process:

a. Difficulty
b. Nonce
c. Merkle root
d. Reward

**Ans: c**

ii. Which of the following parameters is/are applicable (meaning the field(s) will be non-empty and hold some values) to Externally owned account (EoA):

a. Nonce
b. Balance
c. CodeHash
d. StorageRoot

**Ans: a,b,d**

iii. Which of the following will invoke an ethereum transaction:

a. Creation of new EOA
b. Addition of New Ethereum Node to the network
c. Transfer of ethers from one EOA to another EOA.
d. Publish a smart contract code to an EVM.

**Ans: c,d**

iv. A genesis block of a bitcoin blockchain contains _____ transaction(s):

a. 0
b. 1
c. 2
d. None of the above

**Ans: b**

v. What is/are the different way(s) to call or deploy an Ethereum smart contract:

a. Using Web3.js
b. Through another smart contract
c. Through Secure shell (ssh)
d. Using Anydesk

**Ans: a,b**

vi. Which of the following entity(s) in EVM is /are used to execute instructions in a smart contract:

a. Stack
b. ROM
c. Account Storage
d. Queue

**Ans: a,b,c**

vii. A nonce in Ethereum refers to:

a. Seed value used for the mining procedure
b. Identifies the serial number of a block
c. Number of transactions carried out from an account
d. The hash value of the transactions in a block

**Ans: a,c**

viii. Which of the following is used to verify the sender of a transaction in blockchain

a. Hash function
b. Nonce
c. Digital signature
d. Merkle tree

**Ans: c**

ix. Decentralization in Blockchains help:

a. Achieve Consensus of the global state
b. Prevent tampering in the transactions
c. In authentication of the legitimate users of the blockchain
d. reduce bandwidth consumption

**Ans: a**

x. 1 Wei = _____ ETH:

a. $10^{18}$
b. $10^{-18}$
c. $10^{8}$
d. $10^{-8}$

**Ans: b**

**Q2.** Consider the following scenario in Ethereum blockchain: [2 + 1 = 3 Marks]
Suppose Alice sends 1 ETH to Bob, which is finally recorded in the blockchain. The base fee of a block is 12 Gwei and priority fee is 1 Gwei. Total units of gas consumed to execute the transaction is 21,000. Calculate the amount of fees Alice has to pay to the block validators to add her transaction in the blockchain. Also, calculate the final amount that Alice will pay if her transaction is successfully recorded in the blockchain.

**Ans: An ETH transfer requires 21,000 units of gas, and the base fee is 12 gwei. Alice includes a tip of 1 gwei.**

**The fee that Alice will pay to the block validators:**

**units of gas used * (priority fee) = 21,000 * 1 = 21000 gwei = 0.000021 ETH---------- 2 Marks**

**The total fee would now be equal to:**

**units of gas used * (base fee + priority fee)**

**where the base fee is a value set by the protocol and the priority fee is a value set by the user as a tip to the validator.**

**i.e. 21,000 * (12 + 1) = 273,000 gwei (0.000273 ETH).-**

**The final amount that Alice will send: 1.000273 ETH.----------(1 Mark)**

**When Alice sends the money, 1.000273 ETH will be deducted from Alice's account. Bob will be credited 1.0000 ETH. The validator receives the tip of 0.000021 ETH. The base fee of 0.000252 ETH is burned.**

**Q3.** Consider a simple bitcoin block diagram given below. [9 Marks]


What does the following parameters mean/represent:

a. Transactions
b. Miner
c. Reward
d. Fees
e. Distance
f. Difficulty
g. Inputs and Outputs
h. Height
i. Distance

**Ans: (a) Transaction:Bitcoin transactions are messages that state the movement of bitcoins from senders to receivers. Transactions are digitally signed using cryptography and sent to the entire Bitcoin network for verification. Transaction information is public and can be found on the blockchain. The history of each and every Bitcoin transaction leads back to the point where the bitcoins were first produced or 'mined.'**

**(b) Miner:A miner in blockchain refers to a person or a group of people who are responsible for validating and processing blockchain transactions. They are also responsible for adding new blocks to the blockchain. Miners use powerful computers to validate and process transactions. These computers are connected to the blockchain network and work together to solve complex mathematical algorithms.**
**(c) Reward:A reward is a form of compensation given to cryptocurrency miners for validating blocks of transactions on a blockchain. The reward consists of two components: (1) The block reward which is a fixed amount (3.125 BTC) given to miners and, (2) The transaction fees represent the fees collected for including the given transactions in a block of the blockchain.**
**(d) Fees: A Bitcoin network fee, also known as a transaction fee, is a small amount of bitcoin paid to incentivize miners to include the transaction in the next block of the blockchain. While the block reward is stable and predictable, fees can fluctuate due to multiple factors, such as increased activity on the network and transaction size.**
**(e) Distance: It represents the time that has elapsed since the block was mined.**
**(f) Difficulty: It represents the difficulty level of the mining process (which involves solving a mathematical puzzle to find the hash value of the block) with a higher value indicating that the process of finding the block hash is very hard, and vice versa.**
**(g) Inputs and Outputs:The input specifies the source of the cryptocurrency (sender) being sent, while the output specifies the destination (receiver). Altogether, these fields represent the various senders and receivers involved over all transactions in the given block.**
**(h) Height:The block height field in Bitcoin indicates the position (sequence num) of a block in the blockchain. It's calculated from the genesis block, which is the first block in the chain.**
**(i) - Same as point no (e)**

**Q4.** Consider a scenario where there is a need to certify the date a document was created or last modified (Reference can be made to the paper by Stuart Haber and Scott Stornetta, "How to timestamp a digital document"). In order to address this, a digital time stamping service (TSS) is implemented which records the date and time the document was received from a client, and in the event of verification, the records can be verified to determine the client's claim. With respect to the above, answer the following questions, [3+2 = 5 Marks]
(a) What are the various issues with the TSS scheme mentioned above. Please explain the issues clearly.
(b) How did Stuart Haber and Stornetta address these issues? Please clearly mention the tools/concepts they used and explain how the particular tool resolved the issues mentioned in the previous question Q4(a).

Ans:

(a) Issues that are seen in the TSS scheme are as follows: (2 Marks)

Privacy: This method compromises the privacy of the document in two ways: a third party could eavesdrop while the document is being transmitted, and after transmission it is available indefinitely to the TSS itself. Thus the client has to worry not only about the security of documents it keeps under its direct control, but also about the security of its documents at the TSS.

Bandwidth and storage: Both the amount of time required to send a document for time-stamping and the amount of storage required at the TSS depend on the length of the document to be time-stamped. Thus the time and expense required to time-stamp a large document might be prohibitive.

Incompetence: The TSS copy of the document could be corrupted in transmission to the TSS, it could be incorrectly time-stamped when it arrives at the TSS, or it could become corrupted or lost altogether at any time while it is stored at the TSS. Any of these occurrences would invalidate the client's time-stamping Claim.

Trust: The fundamental problem remains: nothing in this scheme prevents the TSS from colluding with a client in order to claim to have time-stamped a document for a date and time different from the actual one.

(b) (3 Marks) With the help of **Hash functions and Digital signatures,** Haber and Stornetta were able to address the issues mentioned above. Hash functions are collision-free compressing bit-strings of arbitrary length to bit-strings of fixed length. Instead of transmitting his document to the TSS, a client will send its hash value h ( z ) = y instead. For the purposes of authentication, time-stamping y is equivalent to time-stamping z. This greatly reduces the bandwidth problem and the storage requirements, and solves the privacy issue as well.

A digital signatures scheme is an algorithm for a party, the signer, to tag messages in a way that uniquely identifies the signer. With a secure signature scheme available, when the TSS receives the hash value, it appends the date and time, then signs this compound document and sends it to the client. By checking the signature, the client is assured that the TSS actually did process the request, that the hash was correctly received, and that the correct time is included. This takes care of the problem of present and future incompetence on the part of the TSS, and reduces the need for the TSS to store records.

To solve the problem of trust, Haber and Stornetta proposed to constrain a centralized but possibly untrustworthy TSS to produce genuine time-stamps, in such a way that fake ones are difficult to produce. **They proposed linking two documents in sequential order while certifying their timestamps.** By including bits from the previous sequence of client requests in the signed certificate of a current client request, it will be clear that the current request was processed after the previous client requests thereby establishing an ordering in timestamps. In this way, a TSS cannot forward-date a document, because the certificate must contain bits from requests that immediately preceded the desired time, and these requests have not been received yet by the TSS. The TSS cannot feasibly back-date a document by preparing a fake time-stamp for an earlier time, because bits from the document in question must be embedded in certificates immediately following that earlier time, yet these certificates have already been issued to the respective clients.

**Q5**. Consider the following output after executing the project in a truffle environment. Answer the questions that follow: [4 Marks]

```
1_initial_migration.js
======================
   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0x5ed06f7718a64e03b57db7c28c6db2c5fcab8d15774e1329dda86a915386ba55
   > Blocks: 0            Seconds: 0
   > contract address:    0xd45559B38a1ecd17f877ad2B2b634F99b3cB1A7a
   > block number:        1
   > block timestamp:     1740117729
   > account:             0xcf278b255204D409Be3836f0527e11E861b304C0
   > balance:             99.99616114
   > gas used:            191943 (0x2edc7)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00383886 ETH
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00383886 ETH

2_deploy_contracts.js
=====================
   Replacing 'Adoption'
   --------------------
   > transaction hash:    0x7ce43e7c462f428c2a4427df8ac6027de42aaf2c0b5844ca8a8e7421f55ad50a
   > Blocks: 0            Seconds: 0
```

```
   > contract address:    0xE16fb1c1AdfdD4299Bc0Fb0653d4584363877595
   > block number:        3
   > block timestamp:     1740117730
   > account:             0xcf278b255204D409Be3836f0527e11E861b304C0
   > balance:             99.99123808
   > gas used:            203815 (0x31c27)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0040763 ETH
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0040763 ETH
Summary
=======
> Total deployments:   2
> Final cost:          0.00791516 ETH
```

a. How many smart contract(s) is/are getting deployed as per the given output. Name the smart contract(s).
b. What is/are the corresponding sender(s) address(es) used to deploy the smart contract(s) you mentioned in response to Q5(a)?.
c. What is/are the fees associated with deploying the smart contract(s) you mentioned in response to Q5(a)?
d. How many transactions is/are created after deploying the smart contracts?

Ans: a. Two. Migration and Adoption are the two smart contracts getting deployed
b. Sender's address for both the smart contracts: `0xcf278b255204D409Be3836f0527e11E861b304C0`
c. Migration: gas used x gas price = 0.00383886 ETH
Adoption: gas used x gas price = 0.0040763 ETH
d. Two transactions, one each for the smart contracts that got deployed.

**Q6.** Consider the web3 deploy script code shown below. Fill in the blanks by specifying what values will the blank spaces - A, B, C, and D, take in the given code. You need not give exact values for the blank spaces but simply mention the name. Example: Block Hash, Transaction Hash, etc. [4 Marks]

```
var storageContract = new web3.eth.Contract(_____A_____);
var storage = storageContract.deploy(_____B_____).send({
     from: _____C_____,
     gas: _____D_____
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' +
contract.transactionHash);
    }
 })
```

Ans:
A: ABI of the smart contract
B: Bytecode after compilation of smart contract
C: Sender's account address
D: Units of gas required to deploy the contract

**Q7.** In the following scenarios, the sequence of steps/activities have been shuffled. Arrange them in the right order. For this question, marking will be done based on the number of activities correctly identified in the right sequence. [2.5 x 2 = 5 Marks]

a. Consider the following set of commands that need to be executed to develop a DApp in a truffle environment. Arrange the sequence in the right order

i. truffle compile
ii. Truffle init
iii. Truffle test
Iv. truffle migrate
V. truffle develop

**Ans: ii, i, v, iv, iii  OR ii, v, i, iv, iii**

b. Consider the following activities that a bitcoin transaction goes through in the P2P network. Arrange the given sequence in the right order
I. transaction is validated by every peer node of the P2P network
Ii. transaction is mined by the miners
Iii. transaction is signed by the sender using its private key
Iv. transaction is broadcast to P2P network
V. transaction appears in the global blockchain

Ans: iii, iv, i, ii, v

**Q8.** Consider the following smart contract which implements the simplest form of a cryptocurrency. The "Coin" contract allows only its creator to create new coins. Anyone can send coins to each other without a need for registering with a username and password, all you need is an Ethereum keypair. Some portions of the smart contract are left blank. **You will have to complete the functions mint() and send()** by writing down the missing parts of the code such that the smart contract functions as a simple cryptocurrency.

N.B.: As the contract is self sufficient, therefore all the necessary state variables, functions, and events have been declared. Hence, do not declare anything new at the contract level, your answer should be restricted to the functions that have been declared in the smart contract. [2 + 2 = 4 Marks]

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.26;


contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping(address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
       // TO DO: Complete the function so that new coins are created

       require(msg.sender == minter);
```

```
        balances[receiver] += amount;

    }

    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);

    // Sends an amount of existing coins
    // from any caller to an address
    // A caller can send coins only if there are sufficient number of coins in
his account
    function send(address receiver, uint amount) public {
      // TO DO: Complete the function so that coins can be transferred from any
caller to a given address

       require(amount <= balances[msg.sender], InsufficientBalance(amount,
       balances[msg.sender]));
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);


    }
}
```