Software Engineering for Business Information Systems
School of Computation, Information and Technology
Technical University of Munich

Eexam
Place student sticker here

Note:

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

Blockchain-based Systems Engineering

Exam:	IN2359 / Endterm	**Date:**	Tuesday 8th August, 2023
Examiner:	Prof. Dr. Florian Matthes	**Time:**	08:00 – 09:30

	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8
I								

Working instructions

- This exam consists of **16 pages** with a total of **8 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 90 credits.
- Detaching pages from the exam is prohibited.
- Allowed resources: none
- Subproblems marked by * can be solved without results of previous subproblems.
- Do not write with red or green colors nor use pencils.
- Physically turn off all electronic devices, put them into your bag and close the bag.

Left room from ________ to ________	/	Early submission at ________

Problem 1 Cryptographic Algorithms and Quantum Computing (7 credits)

0
1
2

a)* Shor's algorithm and extensions to it will break some cryptographic mechanisms that are important to a blockchain's operation as soon as quantum computing hardware advances. Name one mechanism that will be broken and one that will not be broken. For each mechanism, name the type of mechanism and the concrete name of an algorithm.

0
1
2

b)* How widely adopted are quantum-safe cryptographic hash functions in blockchain systems and what is the key reason for the amount of current usage?

0
1
2
3

c)* Alice and Bob create Bitcoin accounts and conduct the following Pay-to-PubKey-Hash (P2PKH) transactions today:

1. Alice receives 6.25 BTC from mining.
2. Alice uses her mining input to send 4 BTC to Bob and the remaining 2.25 BTC back to herself.

Alice and Bob issue no further transactions. A few years in the future, Chuck has access to a powerful quantum computer and tries to steal funds from Bitcoin users. Assume the Bitcoin protocol has not changed. Can Chuck steal coins from Alice or from Bob? For each of them, explain your answer with regard to quantum computing.

Problem 2 Permissionless Consensus (13 credits)

The trade-off between *safety* and *liveness* in consensus mechanisms during an asynchronous period (i.e., when there is an unknown latency for message delivery) can be seen as analogous to the famous Consistency-Availability-Partition (CAP) theorem from distributed computing. The CAP theorem proves that a system cannot achieve *consistency* ($\sim$ *safety*) and *availability* ($\sim$ *liveness*) at the same time when there is a network partition.

a)* Name which CAP property should be preferred during a network partition for the following systems and briefly explain why.

0 1 2 3

1. System: **Search Engine**, State: **Search Results**, Input: **Search Query**
2. System: **Bank**, State: **Account Balances**, Input: **Transaction**

b)* Out of the two consensus mechanisms we learned in class, name the one which is susceptible to forks in an asynchronous period and explain the reason with respect to the CAP property that causes it.

0 1 2

c)* Making a permissionned blockchain permissionless is more complex than letting everyone join the network. Let's assume you have established a permissioned blockchain that has the same consensus mechanism as Bitcoin and uses Round-Robin to select the next block proposer. Now, you want to convert it to a permissionless one:

0 1 2

1. If you open up the registry for being included in Round-Robin to the public, what fundamental problem arises?
2. Name the mechanism required to convert it to a permissionless blockchain while avoiding the problem discussed in point *1*.
3. Give an example of such mechanisms.

In September 2022, Ethereum went through one of the most anticipated upgrades in the crypto community, namely, “The Merge”. With the merge, Ethereum transitioned from a Proof-of-Work (PoW) blockchain to a Proof-of-Stake (PoS) one, consuming 99.95% less electricity.

0 1 2

d)* Why is this upgrade known as a merge and not just a transition to PoS? Briefly explain it concerning the changing architecture.

0 1

e)* It is known that PoS blockchains may fail if there are not enough consensus participants staked in. How did Ethereum address this issue?

0 1

f)* Although Algorand is also a PoS blockchain, it is not endangered by the attack vectors that block building following a deterministic schedule brings (unlike Ethereum). What is the name of the mechanism/technology that provides this? Name of the Sybil-control mechanism is not sufficient!

0 1 2

g)* The Pure-Proof-of-Stake (PPoS) mechanism of Algorand does not constraint the set of accounts that can participate in consensus, and any account with a minimum balance of 0.1 ALGO ($\sim$ \$0.01) can run a participation node. Considering that joining consensus is rather cheap (e.g., compared to staking 32 ETH on Ethereum), briefly explain how Algorand avoids Sybil attacks.

Problem 3 Scaling Bitcoin (13 credits)

One of the aspects of the blockchain trilemma is *scalability* which concerns with the transaction throughput of a blockchain. As Bitcoin suffers from low throughput, it adopted the SegWit soft fork, which enabled placing more transactions into a block without changing the block size limit. SegWit achieved this by introducing *weight units* (WU), and limiting blocks to 4,000,000 WU instead of 1MB.

Alice is a Bitcoin miner who adopted the SegWit upgrade. In Table 3.1, pending transactions in Alice's mempool are listed. Complete the following exercises using this table.

ID	Size (Bytes)	Weight (WU)	Witness*	$\sum$ *Input*(*BTC*)	$\sum$ *Output*(*BTC*)
aa12...	350	α	Yes	0.27	0.21
fb07...	700	β	No	9.25	9.00
6be6...	Ω	3,600	No	5.63	5.33
55cc...	Δ	700	Yes	1.20	0.10

*Witness is 100 Bytes and already included in the Size of relevant transactions.

Table 3.1: Pending Transactions in the Mempool of Alice

a)* Calculate the values of α, β, Ω, and Δ. Only the correct results will be considered.

0 1 2 3 4

α :

β :

Ω :

Δ :

b) Alice attempts to build the next block and she has 5,000 WU space left that she can fill. Assuming that she is economically rational, write the IDs of transactions that Alice will include in her block and calculate the total transaction fee she will earn just from these transactions. Exceeding the available block space will lead to 0 points.

0 1 2 3 4 5

c)* Bob is a friend of Alice who is also doing Bitcoin mining. Unlike Alice, Bob did not adopt the SegWit upgrade. Whenever a new block is mined, Alice and Bob compare the size of the block (in MB) they received. Name the two possible scenarios in terms of size comparison [<, >, =] and briefly explain how they can happen.

0 1 2 3 4

Problem 4 Ethereum Block Header (10 credits)

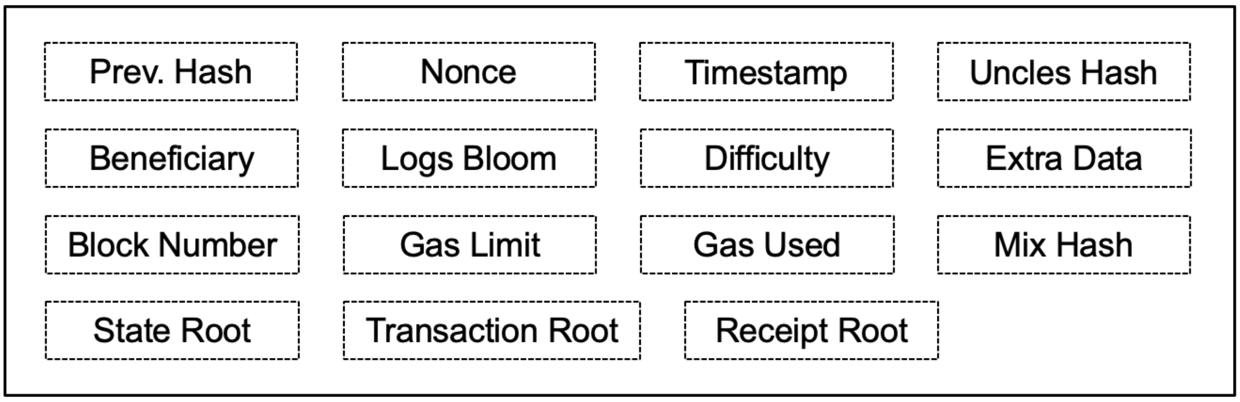

Figure 4.1: Top-Level View of a Proof-of-Work Ethereum Block Header

a)* Figure 4.1 shows the top-level block header when Ethereum was still running Proof-of-Work. After “The Merge”, certain fields in the header became useless. Name two of these fields.

b)* Due to the advanced state structure, Ethereum can be considered more as a distributed state machine than a simple distributed ledger. To maintain the state, Ethereum utilizes Merkle-Patricia Tries (MPT) and stores the roots of the MPTs on the block header.
For each use case listed below, **list the set of MPT roots needed**. If you list more than the necessary number of roots, we will only consider your first *N* roots which match the solution size.
Note: MPT roots are not limited to the ones listed in Figure 4.1.

1. Calculating the transaction fee paid by a certain transaction
2. Checking if any ERC-20 token was transferred (without going through the transactions)
3. Checking Ether and ERC-20 token balances of an address
4. Checking the liquidity available in an on-chain Automated Market Maker (AMM) pool

c)* The London Hard Fork in 2021 relaxed the maximum Gas Limit field from 15M to 30M gas, while the target gas limit remained at 15M. This change was a part of the newly introduced transaction fee mechanism EIP-1559. Briefly explain how these values are relevant for EIP-1559.

Problem 5 Ethereum Transactions (12 credits)

Carrie has the following transaction which is included in the last confirmed block. Table 5.1 displays the details of this transaction and its input data (calldata).

Sender	Recipient	Nonce	Value	Gas Limit	Gas Used	Base Fee	Tip
`0xaa22...`	`0xdd44...`	51	0 ETH	500,000	420,550	18 Gwei	0.2 Gwei

Calldata

[0]: `0x23B872DD`
[1]: `000000000000000000000000CC6295B8B5E9DAD2D3BA15EF4CD4C47A4C0AE2FA`
[2]: `000000000000000000000000BB8F95DBC639621DBAF48A472AE8FCE0F6F56A6E`
[3]: `00000000000000000000000000000000000000000000000000000000000000FF`

Table 5.1: Details of Carrie's Transaction

Function Signature	**Keccak Hash**
`allowance(address owner, address spender)`	`dd62ed3e90e97b3d417db9c0c7522647811bafca5afc6694f143588d255fdfb4`
`approve(address spender, uint256 value)`	`095ea7b334ae44009aa867bfb386f5c3b4b443ac6f0ee573fa91c4608fbadfba`
`burn(address account, uint256 value)`	`9dc29fac0ba6d4fc521c69c2b0c636d612e3343bc39ed934429b8876b0d12cba`
`mint(address account, uint256 value)`	`16267c0d056579491157dd84f60a063a564dc2160bb7ebeb4396e7481bc609ea`
`transfer(address to, uint256 value)`	`a9059cbb2ab09eb219583f4a59a5d0623ade346d962bcd4e46b11da047c9049b`
`transferFrom(address from, address to, uint256 value)`	`23b872dd7302113369cda2901243429419bec145408fa8b352b3dd92b66c680b`
`update(address from, address to, uint256 value)`	`8ce516dafdf7e6222dd04731d0a2cd3b2d9f3c9c62b3bf2c97db3b87615b98b8`

Table 5.2: Function Signature Lookup Table

a)* Use the provided function signature lookup table to decode the calldata of Carrie's transaction in Table 5.1. **Specify the function signature and arguments with their value**.
Note: Hex bytes you skip are assumed to be 0. The least significant byte is the right-most one.

b) Carrie's transaction is executing a call to an ERC-20 contract. For this transaction to not revert, another specific call to a function of the same contract must have **already been made**. Use the lookup table and fill in the calldata of **that call** to the function.
Note: Fields below are provided for your convenience. It does not mean all have to be filled.

[0]:

[1]:

[2]:

[3]:

0
1
2

c)* Calculate the fee from Carrie’s transaction that the proposer of the block got to keep.
Note: It is accepted if you only write the closed form of the equation with the correct numbers.

0
1
2

d)* Briefly explain what happens to the fee that the validator does not get to keep and name the macroeconomic influence of this action on the Ethereum blockchain.

Problem 6 **Ethereum Smart Contracts - Solidity (15 credits)**

Alice and Bob are founding a web3 startup. They decide to keep their company funds in Ethereum. To safeguard the funds from a single key compromise and to ensure agreement on company spendings, they decide to implement their own multi-signature wallet in the form of a smart contract. They decide on the following requirements for the contract:

- Alice and Bob are owners of the wallet and both of them need to confirm transactions before they become executable.
- The wallet contract can interact with EOAs or other smart contracts.
- Owners of the wallet can submit transactions they would like to propose.
- Owners of the wallet can confirm a submitted transaction. Submitting a transaction does not automatically confirm it.
- After a proposed transaction has gained the necessary confirmations, any owner can execute the transaction. Executing a transaction makes the wallet contract issue a corresponding message. Each transaction may only be executed once.
- All interactions with the contract should emit fitting events to make monitoring the wallet easy.
- Any illegal interaction or state should lead to a revert.

You are allowed to use all functionality that Solidity provides. As you can see on the next page, the smart contract assumes a current version of Solidity. This Solidity version includes SafeMath and thus you do **not** need to account for over-/underflow of variables.
Some of the following code snippets may be useful:

- Sender of the transaction: `msg.sender`
- Amount sent with the transaction: `msg.value`
- Enforcing conditions: `require(...)`
- Transfer assets: `recipient.transfer(...)`
- Unit literals: ether, finney, wei
- Casting arbitrary data to uint: `uint(...)`
- Casting an address `a` to a payable address: `payable(a)`
- Calling a function of an address `a`: `a.call{value: msg.value}(<byte_data>)`
- Empty address: `address(0)`
- Returns Ether balance of the contract: `address(this).balance`
- Emit an event: `emit <event_object>`

Convert the instructions on the previous page into code to complete this contract [1].

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract MultiSigWallet {
    event Deposit(address indexed sender, uint amount, uint balance);
    event SubmitTransaction(
        address indexed owner,
        uint indexed txIndex,
        address indexed to,
        uint value,
        bytes data
    );
    event ConfirmTransaction(address indexed owner, uint indexed txIndex);
    event ExecuteTransaction(address indexed owner, uint indexed txIndex);

    address[] public owners;
    mapping(address => bool) public isOwner;
    uint public numConfirmationsRequired;

    struct Transaction {
        address to;
        uint value;
        bytes data;
        bool executed;
        uint numConfirmations;
    }

    mapping(uint => mapping(address => bool)) public isConfirmed;

    Transaction[] public transactions;
```

0 1 2 3 4 5 6

a)

```
    // modifiers
```

```
    constructor(address[] memory _owners) {
        for (uint i = 0; i < _owners.length; i++) {
            address owner = _owners[i];
            isOwner[owner] = true;
            owners.push(owner);
        }
        numConfirmationsRequired = 2;
    }

    receive() external payable {
        emit Deposit(msg.sender, msg.value, address(this).balance);
    }

    function getOwners() public view returns (address[] memory) {
        return owners;
    }

    function getTransactionCount() public view returns (uint) {
        return transactions.length;
    }
```

[1] This contract was freely adapted from `https://solidity-by-example.org/app/multi-sig-wallet/`

```
    function submitTransaction(
        address _to,
        uint _value,
        bytes memory _data
    ) public onlyOwner {
        uint txIndex = transactions.length;

        transactions.push(
            Transaction({
                to: _to,
                value: _value,
                data: _data,
                executed: false,
                numConfirmations: 0
            })
        );

        emit SubmitTransaction(msg.sender, txIndex, _to, _value, _data);
    }

    function confirmTransaction(
        uint _txIndex
```

b)

```
    ) public onlyOwner txExists(_txIndex) notExecuted(_txIndex) notConfirmed(_txIndex) {
```

0 1 2 3

```
    }

    function executeTransaction(
        uint _txIndex
```

c)

```
    ) public onlyOwner txExists(_txIndex) notExecuted(_txIndex) {
```

0 1 2 3 4 5 6

```
    }

} // end of the contract
```

Problem 7 Self-Sovereign Identity built on Tezos (12 credits)

0 1 2

a)* Ethereum adopts soft and hard forks for realizing upgrades. How does Tezos realize such upgrades and how does it prevent Sybil attacks?

0 1 2

b)* Starting a blockchain with Proof of Stake is difficult. Why is that and how did Tezos overcome this problem?

0 1

c)* What is it about Tezos smart contracts that differentiate them from Ethereum smart contracts and makes them desirable for critical applications such as Verifiable Data Registries?

0 1 2

d)* Verifiable Data Registries for SSI can take many different forms. An issuer could sign the relevant data and host it on a web server or on a distributed file system. What advantages does a blockchain like Tezos have over that? Name and briefly explain two.

For the following questions, consider this implementation of a Tezos smart contract written in JSLigo, a syntax similar to JavaScript:

```
type credential_status = ["Active"] | ["Revoked"];

type credential_log_entry = {issuer: address, issuance_time: timestamp, status: credential_status};

type parameter =
| ["Add_Trusted_Issuer", address]
| ["Remove_Trusted_Issuer", address]
| ["Log_Issuance", string]
| ["Revoke_Issuance", string];

// smart contract data storage
type storage = {
    owner: address,
    trusted_issuers: set<address>,
    log: map<string, credential_log_entry>
};

type return_ = [list<operation>, storage];

const add_trusted_issuer = (acc : address, store : storage) : return_ => {
    if (Tezos.get_sender() != store.owner) { return failwith("Access denied."); }
    else { return [list([]), {...store, trusted_issuers: Set.add(acc, store.trusted_issuers)}]; };
}

const remove_trusted_issuer = (acc : address, store : storage) : return_ => {
    if (Tezos.get_sender() != store.owner) { return failwith("Access denied."); }
    else { return [list([]), {...store, trusted_issuers: Set.remove(acc, store.trusted_issuers)}]; };
}
```

```
const log_issuance = (cred_hash: string, store : storage) : return_ => {
    const entry : credential_log_entry = {
        issuer: Tezos.get_sender(),
        issuance_time: Tezos.get_now(),
        status: Active()
    };

    return [list([]), {...store, log: Map.add(cred_hash, entry, store.log)}];
}

const revoke_issuance = (hash: string, store : storage) : return_ => {
    return match(Map.find_opt(hash, store.log), {
        Some: (entry:credential_log_entry) => {
            if (Tezos.get_sender() != entry.issuer) { return failwith("Access denied."); }
            else {
                const e : credential_log_entry = {...entry, status: Revoked()};
                return [list([]), {...store, log: Map.update(hash, Some(e), store.log)}];
            };
        },
        None: () => failwith("No issuance found.")
    });
}

const main = (action: parameter, store: storage): return_ =>
    match(action, {
        Add_Trusted_Issuer: a => add_trusted_issuer(a, store),
        Remove_Trusted_Issuer: a => remove_trusted_issuer(a, store),
        Log_Issuance: h => log_issuance(h, store),
        Revoke_Issuance: h => revoke_issuance(h, store)
    });
```

e)* What Verifiable Data Registry functionalities that we talked about in the lecture does it provide? Name and briefly explain them.

0 1 2 3

f) The `log_issuance` method does not have any access control. Explain why that is not necessary to the functionality and why spam is not a concern as well.

0 1 2

Problem 8 **True / False (8 credits)**

Check whether the following statements are **True** (T) or **False** (F). If they are false, write a correction to the solution box. Use the ID number to refer to a statement. Negating the statement is not accepted as a correct answer.

ID	Statement	T	F
1	Ethereum smart contract addresses are hashes of public keys.		
2	Every EIP-1559 transaction in a block has the same base fee.		
3	The prominence of MEV grows with the expanding DeFi ecosystem.		
4	In Constant Product Market Makers, liquidity addition changes the price.		
5	Hyperledger Fabric (HLF) is designed to be used in permissioned blockchain systems.		
6	HLF requires the selection of a Byzantine-fault tolerant ordering service implementation.		
7	HLF peers cannot be endorsing and committing peers at the same time.		
8	Each HLF Channel is a separate blockchain.		

0
1
2
3
4
5
6
7
8

Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.