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 23rd July, 2024 |
| **Examiner:** | Prof. Dr. Florian Matthes | **Time:** | 08:30 – 10:00 |

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

## Working instructions

- This exam consists of **20 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 Basics (9 credits)

Alice runs a faucet for an Ethereum testnet and has experienced some recent denial-of-service attacks where bots drained her faucet's funds. Wanting to secure her faucet with her knowledge of cryptography, she comes up with a mechanism:
For every funding request made, a puzzle has to be solved first by the submitter to prevent spam. This puzzle consists of a challenge $c$ and a hash function $h$. The faucet generates a $c$ for every request. A solution $x$ to the puzzle has to fulfill the following condition:

$$h(c||x) = 42 \tag{1.1}$$

0 1 2 a)* Is there a size limit for the solution $x$? Name the property of hash functions that influences your answer.

0 1 b)* When choosing a hash function $h$, what important property from the lecture should Alice look for to make this mechanism work?

0 1 c)* Name a concrete hash function that could be used as $h$ to make the mechanism work.

0 1 2 d)* Assuming Alice does not re-use challenges, does it matter how exactly she generates $c$. Argue.

e)* The way the puzzle is formulated right now likely makes it too hard to solve. Suggest an easy way to lower the difficulty, that preserves the usefulness of the mechanism.

0
1

f)* If an attacker had access to a quantum computer, could he - most likely - break Alice's mechanism? Explain your reasoning.

0
1
2

## Problem 2 **Bitcoin Transactions (11 credits)**

For the following questions, examine the Bitcoin transactions shown in Figure 2.1 involving Alice, Bob, Carol, and David.

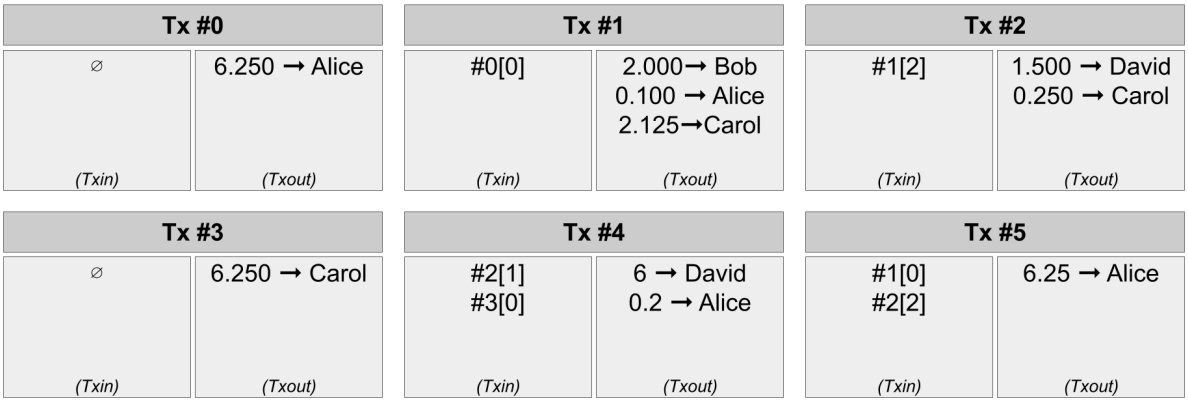



Figure 2.1: Representation of Bitcoin Transactions

0 1 2

a)* Calculate the balance for each entity (Alice, Bob, Carol, and David) after `Tx#2`.

0 1

b)* Identify which transaction(s), if any, is(are) **not valid**.

0 1

c) What is Carol's balance after all **valid** transactions?

0 1 2

d) The miner of the block containing `Tx#4` chose to include only this single transaction from all available transactions in the mempool. How much does the miner earn in **total** for mining this block?

The following itemized list presents the hexadecimal format of a legacy Bitcoin transaction serialization for `Tx#2`. Each line represents a distinct segment of the transaction data in the order they appear in the serialized format, with longer segments wrapped across multiple lines.

- 01000000
- 01
- 04dde43b0e4724f1e3b45782a9bfbcc91ea764c7cb1c245fba1fefa175c3a5d0
- 02000000
- 6a
- 4730440220519f7867349790ee441e83e545afbd25b954a34e0733cd4da3b5f1e5588625050220166730d053c3672973bcb2bb1a977b747837023b647e3af2ac9c15728b0681da01210236ccb7ee3a9f15417f384a05870c4fd86a8727eab7316f1449a0b9e65bfd90d
- ffffffff
- 02
- 5d36010000000000
- 19
- 76a91478364a559841329304188cd791ad9dabbb2a3fdb88ac
- 605b030000000000
- 19
- 76a914064e0aa817486573f4c2de09f927697e1e6f233f88ac
- 00000000

The transaction structure begins with the first 4 bytes (*01000000*) representing the **transaction format version**, which is version 1 in this case. This is followed by a single byte indicating the **In-counter**, which specifies the number of inputs. The subsequent lines contain the input transaction data. After the input data, there is another single byte serving as the **Out-counter**, denoting the number of outputs. The next lines contain the output transaction data. Finally, the structure concludes with the last 4 bytes representing the **Lock Time**. Analyze this transaction data to answer the following questions:

e)* How many scriptPubKeys does the serialized bitcoin transaction of `Tx#2` contain?

0 1

f)* Interpret the Lock Time value represented by the last 4 bytes of the serialized transaction data.

0 1

g)* Explain why the number of confirmations is not specified in the serialized transaction bytes.

0 1

h)* Identify two specific byte-segments in the serialized transaction data that would allow you to distinguish a Segwit transaction from this legacy Bitcoin transaction format, and briefly explain why.

0 1 2

## Problem 3 Bitcoin Consensus (10 credits)

The screenshot in Figure 3.1 is taken from `blockchain.com`, showing the change in Bitcoin difficulty and hash rate over six months.

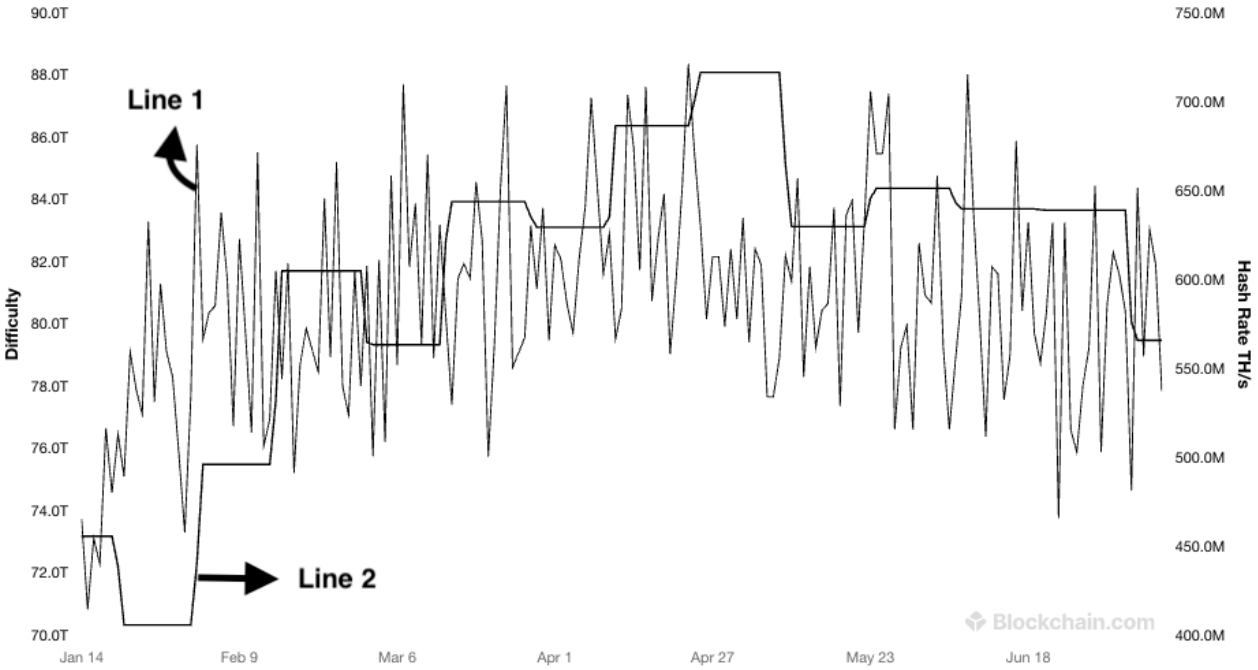



Figure 3.1: Bitcoin Difficulty and Hash Rate

0 1 2

a)* Which plot line most likely represents the Bitcoin **difficulty**? Justify your answer.

0 1

b)* Briefly argue why the two plot lines change similarly.

c)* According to the same website, after April 20, 2024, a significant drop in the daily miner revenue was observed. Beyond the Bitcoin price changes, what event could have affected this drop and why?

0 1 2

Bitcoin's difficulty adjustment mechanism is required for a *selfish-miner* to be successful, making the network believe that the total hash rate is smaller than it truly is. One possible scenario in selfish mining is that an honest miner and a selfish miner propose blocks **simultaneously**.

d)* Why would control over the gossip network help the attacker in this scenario?

0 1

e)* As both the honest and selfish miners' blocks would have the same height, nodes will determine which block to accept based on Bitcoin's consensus rules. Name Bitcoin's **consensus mechanism** and explain how it determines the accepted block in this scenario.

0 1 2

f)* What do we name the block proposed by the honest miner if the network accepts the selfish miner's block? What happens to the transactions in this block?

0 1 2

## Problem 4 Ethereum Transactions (13 credits)

In Table 4.1, you can see the details of all transactions included in the last Ethereum block. The ID field represents unique identifiers but has **no significance in the order of transactions** in which they were executed. Basefee is not shown as it's the same for all transactions.

| ID | Sender | Recipient | Nonce | Value | Gas Limit | Gas Used | Priority Fee | Data |
|---|---|---|---|---|---|---|---|---|
| [1] | `John` | `AuctionHouse` | 11 | 0 ETH | 200,000 | 120,500 | 1.4 Gwei | `setBid(3)` |
| [2] | `John` | `AuctionHouse` | 10 | 0 ETH | 200,000 | 120,500 | 1.2 Gwei | `setBid(5)` |
| [3] | `Eric` | `Paul` | 8 | 1.5 ETH | 12,000 | 12,000 | 1 Gwei | - |
| [4] | `Bob` | `Paul` | 44 | 2.3 ETH | 50,000 | 21,000 | 2 Gwei | - |
| [5] | `George` | `Uniswap` | 90 | 0 ETH | 350,000 | 220,450 | 1.5 Gwei | `swap(10,WETH)` |
| [6] | `Dave` | `Uniswap` | 24 | 0 ETH | 300,000 | 255,140 | 1.1 Gwei | `addLiquidity(..,..)` |

Table 4.1: Transactions in the last block

0 1

a)* Write the IDs of the first **two** transactions in the order they would be included in the block, assuming the validator who proposed the block is economically rational. Only the correct ID in the correct order will be considered for credit.

0 1 2

b)* What would be the final **bid** value of `John`? Explain why.
**Note**: The `setBid()` function sets the bid value of the sender to the given parameter.

0 1 2

c)* How would the ETH balance of `Paul` change? Explain why.

0 1 2

d)* How would the basefee change after this block shown in Table 4.1? Explain why.

Assume the `Uniswap` contract is an Automated Market Maker (AMM) and the `swap()` function swaps the provided amount of specified tokens with the other token in the pair according to the Constant Product formula. The contract was in the following state **before** the last block shown in Table 4.1:

```
swap (tokenInputAmount, tokenInput) // Function Signature
Token USDC reserves: 1000
Token WETH reserves: 10
```

e)* What would be the final state of the token reserves after this block shown in Table 4.1? Ignore the fees. 0 1 2

f)* How would the USDC balance of `Emily` change after this block shown in Table 4.1? 0 1

g)* Assume transaction [5] is included ten blocks later than the current block. Now, it could be possible that `Emily` receives a different amount of USDC tokens. Name this concept in AMMs and write a function call to `Uniswap`, which would cause `Emily` to get **less** USDC tokens. 0 1 2

h)* What is the fundamental difference of transaction [6] to [5] regarding the impact on the Constant Product formula of the `Uniswap` contract? 0 1

## Problem 5 Blockchain Oracles (12 credits)

Blockchain networks such as Ethereum are isolated from the external world. However, decentralized applications on them, such as financial services, require accurate price data. Oracles fill this gap by providing off-chain information to on-chain smart contracts. In this exercise, you are tasked with examining an Oracle service for feeding price data into Ethereum.

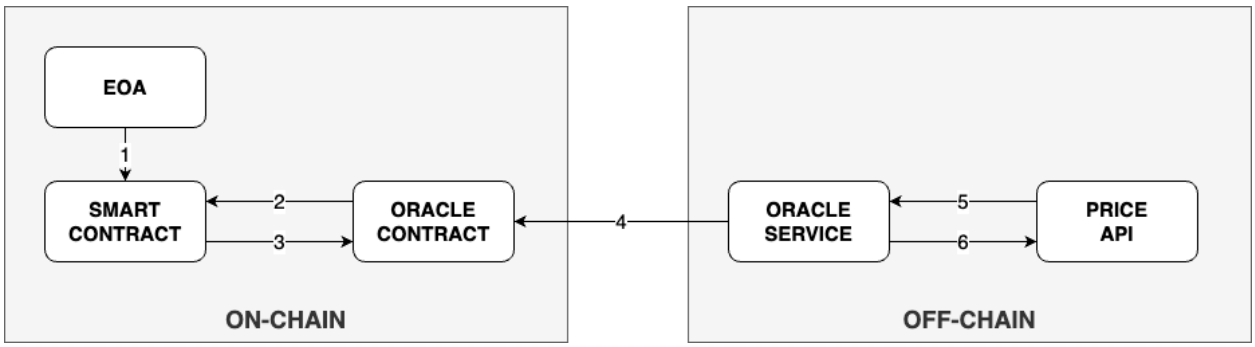



Figure 5.1: Oracle Architecture

In the initial design, the Oracle service updates the price data at every block, and every user call reads the latest available price. Figure 5.1 shows the overall architecture of this design. The calls are indexed, although they do not necessarily follow an order.

0 1 a)* Of the two oracle designs we learned in the course, which one is this oracle service most similar to?

0 1 b)* List the indexes of the *N* calls, which will directly incur an on-chain cost for the **Oracle service provider**. If you list more than necessary, only the first *N* answers will be considered.

0 1 2 c)* List the indexes of the *N* calls, which are Ethereum **messages**. If you list more than necessary, only the first *N* answers will be considered.

0 1 2 3 d)* Calculate the following metrics after the service is run for **10 minutes**. Assume the listed parameters:

- Ethereum Proof-of-Stake block time and EIP-1559 are active
- Gas Usage: 120,000
- Basefee: 30 Gwei (We consider the basefee constant for simplicity)
- Priority Fee: 1 Gwei

1. Cost per oracle service provider transaction:
2. Total burnt amount according to EIP-1559 rules:

The Oracle service provider complains that the cost is too high and there is not enough demand for the service. Thus, they decide to update the price only after a user requests it and only if the last update is older than five blocks. Now, the user will see the last updated price until the new price is available. Further, the Oracle provider wants to track who makes the requests for auditing purposes. As simply checking the transactions included in the block is not efficient enough, they need to use another mechanism.

e)* Answer the following questions regarding the new design:

0 1 2 3

1. Which fundamental Solidity feature can the Oracle service provider utilize to track user requests?
2. Which keyword is needed to ensure the requester information can be efficiently searched off-chain?
3. Which Merkle-Patricia Trie (MPT) root from the Ethereum block header is specifically needed for this mechanism?

f)* Briefly argue why user transactions spend more gas in the new design compared to the initial design.

0 1

While this new design is more cost-effective for the Oracle provider, it may return a stale price for the users until the next update. To address this issue, the Oracle service provider implements a mechanism called *Honest Frontrunner*, which monitors the Ethereum public mempool and updates the price on the contract before the user request is executed.

g)* How can the service provider ensure the Honest Frontrunner transaction gets executed before the user transaction?

0 1

## Problem 6 Ethereum Smart Contracts - Solidity (17 credits)

Alice wants to sell her bicycle to one of her fellow students at TUM. Since she attended BBSE, she figured she might get the most money by auctioning it using a smart contract on the Ethereum Blockchain. The transfer of ownership of the bicycle will happen after the auction ends, so it can be ignored in the smart contract logic.

Implement the contract using current Solidity syntax (version `^0.8.26`). This version includes SafeMath, eliminating the need to manually handle integer overflow/underflow. However, older syntax will be accepted if used correctly.

The provided custom errors in the contract template serve as guidelines for potential edge cases and conditions to check in your implementation. Use these errors by reverting when appropriate to ensure robust contract behavior.

Key Solidity features you may find useful:

- Access the transaction sender: `msg.sender`
- Access the transaction value: `msg.value`
- Transfer Ether: `recipient.transfer(...)`
- Emit events: `emit EventName(...)`
- Revert with custom errors: `revert ErrorName();`

Complete the auction logic in the provided contract[1] template. Pay close attention to the TODO comments for specific function requirements.

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

contract BicycleAuction {
    address payable public beneficiary;

    uint public auctionEndTime; // Times are absolute unix timestamps (seconds since 1970-01-01)

    address public highestBidder;
    uint public highestBid;

    mapping(address => uint) pendingReturns; // Allowed withdrawals of previous bids

    bool is_auction_over;

    event HighestBidIncreased(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    error AuctionAlreadyEnded();
    error BidNotHighEnough(uint highestBid);
    error AuctionNotYetEnded();
    error AuctionEndAlreadyCalled();

    constructor(uint biddingTime, address payable beneficiaryAddress){
        beneficiary = beneficiaryAddress;
        auctionEndTime = block.timestamp + biddingTime;
        highestBidder = beneficiaryAddress; // Add a 0 bid for the auctioneer
        highestBid = 0;
    }
```

[1] This contract is adapted from the Solidity documentation example at `https://docs.soliditylang.org/en/v0.8.26/solidity-by-example.html`

```
    // TODO: Implement the bid function to place bids on the auction. Emit the HighestBidIncreased event with
    //       bidder information and the amount.
```
a) `function bid() external payable {`

0 1 2 3 4 5

```
    }
```

```
    // TODO: Withdraw function: Implement the withdraw function to allow users to withdraw their previous bids
    //       if they are no longer the highest bidder. Be cautious of re-entrancy attacks!
```
b) `function withdraw() external returns (bool) {`

0 1 2 3

```
    }
```

```
    // TODO: Finalize Auction function: Implement the finalizeAuction function to conclude the auction.
    //       The auction end logic is executed only once. Emit the AuctionEnded event with the winner and amount.
```
c) `function finalizeAuction() external {`

0 1 2 3 4

```
    }

} // end of the contract
```

Alice's current auction design for selling her bicycle has a specified ending time, which is leading to last-minute bid surges and potential network congestion as students rush to outbid each other at the end. To address these issues, Alice is considering implementing a **blind auction**.
In a blind auction, bidders submit sealed bids without knowing others' bid amounts. This approach aims to prevent last-minute bidding wars and ensure a fairer process.

0 1

d)* Which cryptographic scheme discussed in the lecture could be used to implement a blind auction?

0 1 2

e) Explain the process of implementing a blind auction using this scheme.

0 1 2

f) What potential weakness exists in this implementation, particularly for an adversary with substantial computational resources, and how could it be addressed?

This page has been left blank for your convenience.

## Problem 7 Self-Sovereign Identity (10 credits)

Self-Sovereign Identity is a new paradigm that seeks to decentralize identity management. Take a look at the data structure in Figure 7.1. The following questions will all reference it.
**Note:** In your answers, you may shorten any identifier as long as it is still easily and uniquely identifiable.

0 1 2 a)* What is the JSON-based data structure in Figure 7.1 and what organization governs this standard?

0 1 b)* What DID Methods are used in the data structure in Figure 7.1?

0 1 c)* What is the general cryptographic concept that the proof section in Figure 7.1 is based on?

0 1 2 d)* Who is the data structure in Figure 7.1 originally given to? List any identifier of that entity you can find.

0 1 2 3 e)* What is the specific data structure in Figure 7.1 trying to say in a way that humans would talk like? Fill in the template below, and be careful to stick to things you can know with certainty from just Figure 7.1.

_____confirms that _____
is _____

0 1 f)* Showing you the data in Figure 7.1 is not a security risk because you can only present it to a relying party in the form of a Verifiable Presentation. Explain how that prevents misuse.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1"],
  "id": "urn:uuid:89581491-c9d6-47d2-bd4b-e606fe6acd70",
  "type": ["VerifiablePresentation"],
  "verifiableCredential": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      {
        "EmailPass": {
          "@context": {
            "@protected": true,
            "@version": 1.1,
            "email": "schema:email",
            "id": "@id",
            "issuedBy": {
              "@context": {
                "@protected": true,
                "@version": 1.1,
                "logo": {
                  "@id": "schema:image",
                  "@type": "@id"
                },
                "name": "schema:name"
              },
              "@id": "schema:issuedBy"
            },
            "schema": "https://schema.org/",
            "type": "@type"
          },
          "@id": "https://github.com/TalaoDAO/context#emailpass"
        }
      }
    ],
    "id": "urn:uuid:c2ceaca0-8e9b-11ee-9aa4-0a5bad1dad00",
    "type": ["VerifiableCredential", "EmailPass"],
    "credentialSubject": {
      "id": "did:key:z6MkkdC46uhBGjMYS2ZDLUwCrTWdaqZdTD3596sN4397oRNd",
      "email": "felix.hoops@tum.de",
      "type": "EmailPass",
      "issuedBy": {
        "name": "Altme"
      }
    },
    "issuer": "did:web:app.altme.io:issuer",
    "issuanceDate": "2023-11-29T09:43:33Z",
    "proof": {
      "type": "Ed25519Signature2018",
      "proofPurpose": "assertionMethod",
      "verificationMethod": "did:web:app.altme.io:issuer#key-1",
      "created": "2023-11-29T09:43:33.482Z",
      "jws":
"eyJhbGciOiJFZERTQSIsImNyaXQiOlsiYjY0Il0sImI2NCI6ZmFsc2V9..wl9s4OXCG5vV_sDvxn0E8DmHqQ482e2BlKy-
sRsIN9WSwO0ZTU3O75wnEl0PtAcwIFPz_3VIlpz9hjJcRUqABA"
    },
    "expirationDate": "2024-11-28T09:43:33.446349Z"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "proofPurpose": "authentication",
    "challenge": "test",
    "verificationMethod":
"did:key:z6MkkdC46uhBGjMYS2ZDLUwCrTWdaqZdTD3596sN4397oRNd#z6MkkdC46uhBGjMYS2ZDLUwCrTWdaqZdTD3596sN4397o
RNd",
    "created": "2023-11-29T14:12:48.142Z",
    "domain": "https://ec80-2003-ee-af45-6c00-e0d1-7850-acea-8745.ngrok-free.app",
    "jws":
"eyJhbGciOiJFZERTQSIsImNyaXQiOlsiYjY0Il0sImI2NCI6ZmFsc2V9..cUfNpVhLFOmBIebiJO345ImTzKN0_G9Al2k8dJx7wcYv
XCfyfWnxFdCGCi13c2tNj6bA-RbzFmo6qrEaQTxtAw"
  },
  "holder": "did:key:z6MkkdC46uhBGjMYS2ZDLUwCrTWdaqZdTD3596sN4397oRNd"
}
```

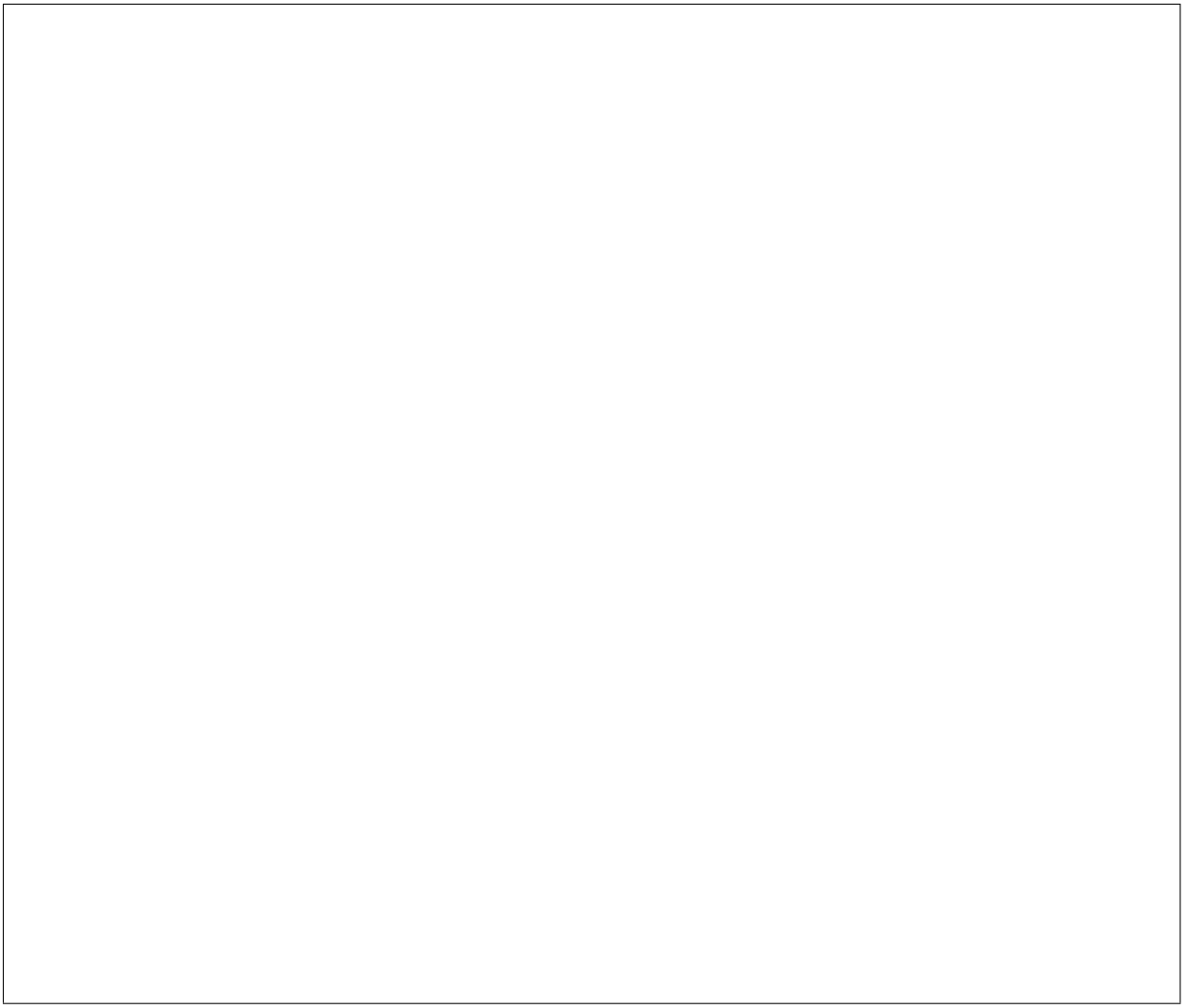

Figure 7.1: Real data captured from an SSI ecosystem.

## Problem 8 True / False (8 credits)

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

| ID | Statement |
|---|---|
| 1 | There is no objectively correct number of highest unconfirmed transaction count in Bitcoin |
| 2 | Bitcoin requires a fork if transactions in the mempool are deleted after a certain time. |
| 3 | EVM refunds the spare gas of Ethereum transactions with too much gas provided. |
| 4 | Hyperledger Fabric (HLF) makes it possible to have private p2p connections within a ledger instance. |
| 5 | HLF requires at least two ordering nodes. |
| 6 | HLF makes use of the Order-Execute strategy to achieve higher throughput than otherwise possible. |
| 7 | ZK Rollups post more data on Ethereum than Optimistic Rollups for a 10,000 transaction batch. |
| 8 | Data consistency and availability are key challenges in implementing sharding. |

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.**