University of Stavanger

Faculty of Science and Technology

Exam in subject:	**DAT650 Blockchain technology**
Date:	**Eksamen, 2nd December 2019**
Duration:	**4 hours**
Allowed remedies:	**None**
The exam consists of:	**11 exercises on 9 pages**
Contact during exam:	**Leander Jehl, tlf. 94120764**
Remarks:	None
Attachments:	

Question 1: Data structures (6%)

(a) (4%) Sketch a Merkle tree with 5 data elements. How is the root computed from the different elements?

Solution:

$$h_{root} = H(H(H(D_1, D_2), H(D_3, D_4)), H(H(D_5, D_5), H(D_5, D_5)))$$

(b) (2%) Let t_1 and t_2 be two transactions included in the Bitcoin Blockchain. Can we deduce which was issued first?

Solution: If t_1 is included in an earlier block than t_2 we can assume that t_1 was issued first.

Question 2: Unspent transaction output (UTXO) (20%)

(a) (6%) What are the different elements of a transaction in Bitcoin? How is a transaction validated?

Solution: Transactions include inputs and outputs. Validation checks that

- Inputs reference previously unspent outputs
- Inputs fulfill condition on outputs (i.e. signature with correct key)
- Sum of outputs is less or equal to inputs.

(b) (4%) How is a coinbase transaction validated?

Solution: Need to check that the output is equal to the block reward plus fees from transactions in the block.

(c) (5%) Ethereum does not use the UTXO model. Therefore every transaction in Ethereum includes a *nonce*. Explain why?

Solution: The nonce prevents replay attacks. A transaction with the same nonce can be executed only once. The next transaction needs a higher nonce.

(d) (5%) An output in bitcoin typically specifies the public key that needs to be used to spend this output. For this there are two variants:

A. The output can include the public key.

B. The output can include the hash of the public key.

What are the advantages of alternative A. for

- the user sending the transaction
- the system (nodes in bitcoin)

Solution: The sender needs to pay a fee for its transaction. If he uses alternative A. his transaction is shorter, and he thus needs to pay a smaller fee.

The network nodes need to store the unspent transaction outputs in memory, until they are spent. Alternative A. gives a smaller output and storing it uses less memory.

Question 3: Proof of work (18%)

(a) (3%) Assume the block delay of a bitcoin like blockchain is 6 minutes. What is the probability that a block is found in the next second?

Solution: $\frac{1}{360}$

(b) (3%) During the last 2016 blocks, one block was found on average every 6 minutes. How should the difficulty d be adjusted, to make sure blocks are found every 10 minutes in the future. *You may use the definition below for help.*

Solution: Set $d \cdot \frac{3}{5}$

Def: *For a hexadecimal number d, the proof-of-work function with difficulty d takes a data item and returns a nonce (random bits) and a hash value:*

$$(h_{PoW}, nonce) = f_{PoW}(Data)$$

The proof of work is valid, *if a)* h_{PoW} *is the hash of the data, concatenated with the nonce*

$$h_{PoW} \stackrel{?}{=} H(Data||nonce))$$

and b) h_{PoW} written as hexadecimal number is smaller than d.

$$h_{PoW} < d$$

(c) (4%) Assume Alice finds a nonce that allows her to solve the PoW puzzle and create a new block for the Bitcoin blockchain. Alice sends this block to the bitcoin network.

Assume further that Bob is the first node in the network to receive Alices block. Can Bob take the Nonce from Alices block and cash in the block reward himself?

Solution: No. To receive the block rewards, Bob needs to include his address in the coinbase transaction. Thus his block will be different from Alices block and he will most likely need a different nonce.

(d) (4%) We said that a PoW function needs to have fast verification and progress freedom. Explain why the bitcoin PoW function has both these properties.

Fast verification Every node in the network needs to verify a solution. Thus verification should be easy, compared with computation.

Progress free The probability to solve the PoW function in the next second, should be independent of how long a process has been trying to solve it.

Solution: The hashing in bitcoin has fast verification, since you can try many nonces to find one that solves the puzzle. To verify, you only have to compute one hash.

It is progress free, since, whether the one nonce solves the puzzle is independent of if another nonce solves it.

(e) (4%) Name a reason to decrease the block interval in bitcoin (currently 10 minutes). What problems does an increased blocksize cause?

Solution: A decreazed block interval means that more transactions can be committed every 10 minutes. However, it will also result in more forks.

Question 4: Forks (4%)

Assign for each of the following whether it is a soft, hard, or soft and hard fork.

- Require that every transaction in Bitcoin pays a fee that lies between 0.00005–0.0001 Bitcoin.
- Increase the maximum block size.

Question 5: 51% Attack (10%)

(a) (4%) What happens in bitcoin, if two transactions spending the same output are issued? Can both transactions be included in a block? How and when can one of those transactions be considered confirmed or committed?

Solution: Only one of the two transactions can be included in one block or chain. But both can be included in different blocks on different forks.

One transaction is considered confirmed, if it is included in a block, and this block was extended by 5 other blocks.

(b) (6%) What is a 51% attack in Bitcoin and how could an attacker perform and benefit from double spending?

Solution: In a 51% attack the attacker controls the majority of the hashing power in the network. He can use this to privately create a chain, or fork that will eventually be the longest chain.

If the attacker creates a secret chain with 6 or more blocks, he can publish this chain and undo a confirmed transaction. This allows to issue a payment, receive goods and later undo the payment. In the case of a double spend, the specific payment cannot be re-added later.

Question 6: Selfish mining (14%)

In this question we consider an example of selfish mining. We assume in the blockchain in Figure 1, an attacker has secretly mined a block b_c.

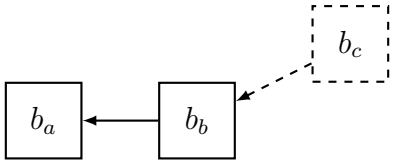

Figure 1: Selfish mining example: Dashed block is mined secretly.

(a) (4%) How will the attacker react if another block is added to the public chain, as shown Figure 2. What are the possible outcomes?

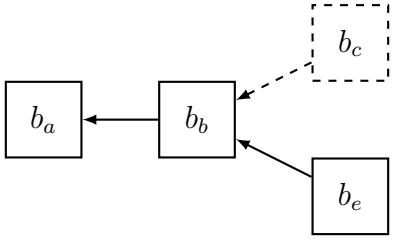

Figure 2: Selfish mining example: Dashed block is mined secretly.

Solution: The attacker will immediately publish block b_c, resulting in a fork. Either b_c or b_e will be extended to become longest chain. If b_e becomes the longest chain, the attacker will loose the block reward for b_c.

(b) (4%) In the Situation from Question 6a) above (Figure 2), how can the attacker benefit, if he additionally performs a forwarding denial attack?

Solution: Using a forwarding denial attack, the attacker can slow down the dissemination of block b_e. This gives a better chance that b_c will be extended.

(c) (3%) Assume the selfish mining attack is done on Ethereum. In the Situation from Question 6a) above (Figure 2), why is the existence of Uncle blocks a benefit for the attacker?

Solution: If b_e becomes part of the longest chain, the attacker will not loose the complete block reward for b_c, since he can still receive the uncle reward.

(d) (3%) Assume the attacker manages to mine another secret block (b_d), before b_e is found, as shown in Figure 3. How will the attacker react in this situation?

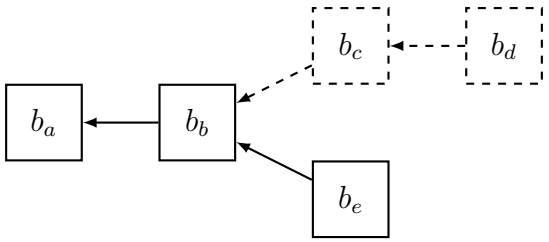

Figure 3: Selfish mining example: Dashed blocks are mined secretly.

Solution: The attacker will immediately publish block b_c and b_d. All honest miners will continue to mine on top of b_d.

Question 7: Proof of Stake (6%)

In PPCoin (Peercoin) a miner identified by *addr*, that has deposited `coin`(*addr*) can supply the current block, if

$$H(\texttt{prevBlockHash}||addr||\texttt{timeinseconds}) < d_0 \cdot \texttt{coin}(addr)$$

- Here d_0 is a base difficulty. The probability that a miner with a specific address *addr* can mine the next block is proportional to `coin`(*addr*).
- `timeinseconds` shows time in seconds. Thus a miner gets a change to submit a solution every second.

Mention **two problems** or possible attacks in PPCoin.

Solution:

Predictability A miner can predict whether he will be able to mine the next block.

PoW next block A miner with sufficient resources can try to tweak the current block, s.t. he will be able to also mine the next block.

Non deciding In case of a fork, a miner does not have to decide on which block he wants to mine. It is feasible to mine of both chains. Thus forks may prevail for long.

History rewrite Theoretically it is feasible to rewrite the complete, or a large part of the history of this chain.

Question 8: GHOST and Longest chain rule (5%)

Figure 4 shows an example of a chain with many forks.

(a) (2%) Which block should be extended according to the longest chain rule?

(b) (3%) Which block should be extended according to the GHOST rule?

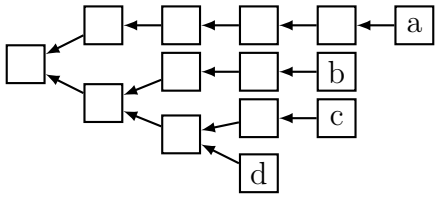

Figure 4: Example chain with many forks.

Question 9: Bitcoin-NG (4%)

Bitcoin-NG defines both key-blocks and micro-blocks. Fees from Micro-blocks are divided 40/60 between the creator of the last and next key block.

(a) (2%) What attack would be possible if the creator of the next key block would not get a part of the fees, e.g. a 100/00 divide?

> **Solution:** A miner could, instead of extending the micro-blocks, extend the last key block. This allows him publish his own micro-blocks, including the transactions.

(b) (2%) What attack would be possible if all fees would go to the next block, i.e. a 00/100 divide?

> **Solution:** Miners would no longer publish microblocks.

Question 10: System models and BFT (4%)

(a) (3%) In an unpermissioned system like Bitcoin, there exists no membership table or similar. In such systems, how can you hold a vote, e.g. if a certain update should be performed? What is needed to win the vote?

> **Solution:** Miners can vote by including a vote in the headers of blocks they publish. You can see, at some point if the majority of blocks vote for or agains the update. To win the vote, you need a majority of the mining power.

(b) (4%) What is the difference between a BFT failure model and a failure model assuming rational/selfish nodes?

> **Solution:** BFT assumes that only few nodes fail, but allows them to fail arbitrarily. Rational assumes that all nodes may fail at once, but only if they are incentivized to do so.

Question 11: Ethereum (6%)

Algorithm 1 below shows solidity code for an Ether King Game. Players can participate in the game by paying 0.5 ether. The 6th, 10th and 20th player will win 2, 3 and 5 ether respectively.

Describe two security vulnerabilities of the code in Algorithm 1.

Algorithm 1 Ether King Game

```
1: pragma solidity =0.5.11
2: contract EtherGame {
3:
4:    uint public payoutMileStone1 = 3 ether;
5:    uint public mileStone1Reward = 2 ether;
6:    uint public payoutMileStone2 = 5 ether;
7:    uint public mileStone2Reward = 3 ether;
8:    uint public finelMileStone = 10 ether;
9:    uint public finalReward = 5 ether;
10:   bool public finished = false;
11:
12:   mapping(address => uint) redeemableEther;
13:   // Users pay 0.5 ether. At specific milestones, credit their accounts.
14:   function play() public payable {
15:      require(msg.value == 0.5 ether);            // each play is 0.5 ether
16:      uint currentBalance = address(this).  balance + msg.value;
17:      // ensure no players after the game has finished
18:      require(!finished);
19:      // if at a milestone, credit the payer's account
20:      if (currentBalance == payoutMileStone1) {
21:         redeemableEther[msg.sender] += mileStone1Reward;
22:      }
23:      else if (currentBalance == payoutMileStone2) {
24:         redeemableEther[msg.sender] += mileStone2Reward;
25:      }
26:      else if (currentBalance == finalMileStone) {
27:         finished = true;
28:         redeemableEther[msg.sender] += finalReward;
29:      }
30:      return;
31:   }
32:
33:   function claimReward() public {
34:      // ensure the game is complete
35:      require(finished);
36:      // ensure there is a reward to give
37:      require(redeemableEther[msg.sender] > 0);
38:      uint transferValue = redeemableEther[msg.sender];
39:      redeemableEther[msg.value] = 0;
40:      msg.sender.send(transferValue);
41:   }
42: }
```