ATTENTION - READ FIRST

Please start a discussion by adding a comment on the side. This allows for a better overview of possible mistakes & changes. Thank you!

Exam 2018

## Exercise 1

1. Key Properties
   1. Preimage resistance: if given h and y with h(x) = y it’s computationally infeasible to find a corresponding x with h(x) = y
   2. 2nd preimage resistance: if given h(x) and x it’s infeasible to find y where h(x) == h(y) and x != y
   3. Collision resistance: if given h(x) it’s infeasible to find two values x and y where h(x) == h(y) and x != y
   4. Hiding (desirable): A hash function h is said to be hiding if a secret value r is chosen randomly 1, then, given h(r||x), it is infeasible to find x.
   5. Properties of hash functions in general
      1. Compression: the results of the hash function have the same bit length
      2. Ease of computation: the result of the hash function must be easy to compute
2. 

## Exercise 2

1. The coinbase transaction contains the mining reward for the miner. It is the first transaction in a block and created by the miner itself (who mined that block).
2. The miner can directly change the nonce, the time and the merkle root hash (so the list of transactions).
   1. In the coinbase transaction the scriptSig field can contain a custom data value defined by the miner
3. Data can also be stored inside of the data field in a coinbase transaction (by a miner) (this stores it directly in the chain). There is also a way of storing data through bitcoin script (*this was not handled in this years lecture in detail*)
4. No, as those TX (inventing BTC) would be denied by the rest of the network
5. TX:

#2[0] | 15 → Bob

#3[0] | 8 → Carol

| 2 → Alice

1. In Bitcoin, either all or none of the coins have to be consumed by another transaction. The address the money is sent back to is called a change address. This enables an efficient verification, as one only has to keep a list of unspent transaction outputs (UTXO). The money is not sent back to the initial accounts address but instead a new change address to allow pseudonymous usage.
2. TxIn
   1. Sig1
   2. PubKey1
   3. Sig1
3. The owner of the pub key which hashed is PubKeyHash1 (Alice)
4. No, as we do not see the creator in the TxOut script, only the allowed spender (PubKeyHash1).

## Exercise 3

1. 1. Every program shares the same computer resources

2. Every usage (computation or storage) costs money

1. Every executed opcode instruction uses a miner’s computational resources

and therefore costs a certain fee (called gas). Each opcode uses a certain amount of gas which may depend on the arguments of the operation, e.g., number of bytes to be allocated.

1. STARTGAS is the maximum amount of gas a transaction issuer (EOA) is ready to pay for
   1. GASPRICE is the price (in eth) of one unit of gas such transaction
2. Transactions can only be sent by externally owned accounts and messages only by smart contract. Transactions are mined into the blockchain, messages are not.
3. Because this would work against the idea of having deterministic and transparent algorithms
4. Is an EOA as the contract code and storage is empty/null
5. You can use an oracle to access data from an arbitrary REST API
6. Answers
   1. A+b might overflow.
   2. Pure functions don’t read or modify state variables.
   3. We can use the safemath library or increase our solidity version > 0.8 which automatically includes safemath
      1. Proposed code snippet:

| uint8 c = a + b; assert(c >= a && c >= b); return c; |
| --- |

1. Answer:

| pragma solidity >=0.8.4;  contract PaySalary {   mapping (address => uint) payments;  constructor() public {}  function paySalary() public payable {  require(msg.value > 2 ether);  payments[msg.sender] += uint(msg.value); }   function hasMadePayment(address sender) public view returns(bool) {  return payments[sender] > 0; }  } |
| --- |

## Exercise 4

1. Channels are own blockchains that have their own ledger. A node can be part of one or multiple channels. Each channel has its own chaincode
2. It can be an endorsing or committing peer.
3. Questions
   1. IOTA ⇒ Not part of this years lecture
   2. No → Chaincode is one or multiple smart contracts combined
   3. No → Committing peers do not need to have the chaincode
   4. Yes
   5. No, each endorsing peer is always a committing peer
   6. IOTA ⇒ Not part of this years lecture
   7. IOTA ⇒ Not part of this years lecture
   8. Transaction Bundle ⇒ IOTA ⇒ Not part of this years lecture

## 

Exam 2019

## Exercise 1

1. This property defines, that when having an x with h(x) = y it is computationally infeasible to find another x’ with x!=x’ and h(x’) = y
2. Bloom Filter ⇒ Skip
3. Bloom Filter ⇒ Skip
4. Alice has to send
   1. Hash 0 (h0)
   2. Hash 1-1 (h1-1)
5. Calculate htx3 = h(tx\_3), then h1 = h(htx3+h1-1) and then h(h0+h1) and check if that result matches the root hash

## Exercise 2

1. Missing fields
   1. 1: Hash of previous block header
   2. 2: Nonce
2. TXs are stored using the hash of the merkle root which covers all TX hashes
3. To keep the network speed (blocks / time) consistent
4. A Txout is used to create coins and transfer them, they are generated in a transaction and exist (until spent) as UTXO. In ScriptPubKey conditions are stored that define who can use / spend the value of the Txout
5. It is used to define the sources of money which are then used in Txouts. It’s stored in Txout as BTC is a transaction based ledger (list of UTXOs)
6. When a Txout is not used / referenced in a Txin yet and this TX was not put into a block yet, it is a UTXO. The memory pool contains all transactions that are awaiting confirmation (not UTXOs!)
7. They are called coinbase transactions and their purpose is to pay the reward to the miner who mined a block (where the coinbase is the first TX)
8. Tx3 is wrong, as the sum of Txout values is more than the sum of Txin values
9. Dave
10. The rest of the coins is the transaction fee
11. Tx5
    1. #2[1] | 12 → Tom
    2. #4[0] | 1 → Alice
12. As the blocks communicate peer-to-peer it can happen that two different solutions are proposed to the network simultaneously. It is resolved once the next block is built on either 3A or 3B, then this will be the longest chain and will persist
13. 3A is called an orphan block. The transactions that are not part of 3B already, so still unconfirmed are back in the mempool
14. Txin Script
    1. Sig1
    2. PubKey1
    3. PubKey1
15. Types: P2PK, P2SH. (Details were not covered in this year's lecture)

## Exercise 3

1. Missing Headers
   1. 1: Value
   2. 2: Max Gas (or Gas limit) = STARTGAS
   3. 3: Used Gas
   4. 4: Gasprice
   5. 5: Data
2. There are EOAs and smart contracts. EOAs can only send transactions and smart contracts can only send messages. Contracts have contract bytecode.
3. 0xdddd is an EOA as the account created a transaction (in 2)
4. That’s transaction 4. Because the transaction already took computation power until its maximum was reached, it’s similar to a car not reaching a destination. The gas is still used.
5. For each TX multiply used gas with gasprice, sum it up and divide it by 10^9 (from gwei to ether)
6. The miner chooses the TX with the highest possible fee (gasprice and max gas) to get the most money out of it. As ETH is capitalistic, so are the miners.
7. As messages do not need to be confirmed as they are only sent by contracts and their origin is always a transaction (that called the contract). In addition, message will not cause any non deterministic behavior and can be reproduced by their origin TX which again is part of a block.
8. 1. Fungible tokens

mostly used to collect money via initial coin offerings (ICOs). Usually, tokens work as a sub-currency of Ethereum and represent a certain asset such as a stock.

2. Non-fungible Tokens - NFTs

NFTs are unique cryptographic tokens that are stored on a blockchain that represent a unique asset like music, art, fashion.

3. Decentralised Autonomous Organizations - DAOS

DAOs are non-hierarchical business models that are collectively owned by a group of members. Think of it as a generalization of multi-signature wallets where all decisions are taken based on members’ votes. Thus, no single member can unilaterally take a decision (e.g., send a transaction). DAOs are useful for starting an organization with people you have limited trust (e.g., only communicating over the internet) since you only need to trust the system design and no other single member.

1. 1. Write the source code of the smart contract in Solidity

2. Compile the source code into EVM bytecode using the solidity compiler

3. Bytecode is sent as a TX to the network

4. The byte code is put into a block and mined

## Exercise 4

1. A design pattern is a reusable solution template for a problem that often occurs.
   1. As smart contracts can not be patched and therefore require higher safety standards it is good to stick to design patterns that ensure safe executions.
2. You can use an oracle to retrieve random numbers externally. The blockhash is an alternate approach to get something random (as this one is defined once the TX is mined and not known before). One can although criticize that the miner can influence that blockhash and therefore might exploit it (MEV).

| modifier onlyOwner() {  require(owner == msg.sender);  \_; }  constructor () public {  owner = msg.sender;  winner = address(0);  pot = uint(0); } function payIn () public payable {  require(msg.value >= 1 ether);  players.push(msg.sender);  pot += uint(msg.value); }  function selectWinner() public onlyOwner {  require(winner == address(0));  uint random\_num = uint(blockhash(block.number - 1));  winner = players[random\_num % players.length]; }  function withdraw() public {  require(msg.sender == winner);  uint value = pot;  pot = 0;  winner = address(0);  players = [];  payable(msg.sender).transfer(value); } |
| --- |

1. One can although criticize that the miner can influence that blockhash and therefore might exploit it (MEV). Also the blockhash and block.number are global variables available to everyone on-chain when calculating the random value (bad practice).
2. Commit & reveal:

Another potential approach is the commit and reveal scheme, which utilizes the hiding property of cryptographic hash functions. It consists of two phases:

1. Users (>1) hash their random number and publish it to the smart contract.

2. Users reveal their random number and the smart contract calculates a random number from these commits.

Oracles:

Oracles are usually used to access data from outside of the blockchain. Additionally, Oracles can provide

random numbers, either from local sources or from services (e.g., WolframAlpha).

## Exercise 5

a) Benefits:

1. usually achieve a much higher throughput rate
2. only involved parties are able to issue transactions and participate in the network
3. They do not depend on a cryptocurrency nor on miner incentives

Disadvantages:

1. The system is more complex
2. It makes the chain again very centralized as there is one Membership Service Provider for authorizations (blockchain governance is centralized)
3. Interoperability between different consortia (*cite from the slides*)

b) By using execution - order - validation, we can ensure that the results of executing the contract in all nodes are the same.

Applications need to follow endorsement policies. If RW-sets differ, the endorsing peers will reject them.

c) 2\* (½)^3 = 1/4

Exam 2020

## Exercise 1

1. The goal of a public blockchain is to maintain transparency to all and allow everyone to participate. An encryption of that data would work directly against that idea.
2. Asymmetric cryptography enables an authentication system with public (“user-identifier”) and private (“user-passwords”) keys. This can be used to sign e.g., Transactions and proof that you are the owner / creator of such transaction ⇒ Digital Signature Scheme
3. Brain-Wallet: Keep a passphrase in your brain (Other options: Hardware Wallet like ledger nano, paper wallet, online wallet)
4. Properties
   1. Compression (each result of h(x) has the same bit length)
      1. h has it, as x mod 5 results in a number always smaller than 5, being always the same decimal amount and therefore the same bit length
   2. Easy to compute
      1. h has it, as x mod 5 is easy to compute as modulo is a basic math operator supported by any modern CPU
5. *Explanation: 12 mod 5 = 2 ⇒ 2nd preimage resistance. Coll. Res. can be the same as it only differs from 2nd preimage res. in what you know already (e.g., x\_1 in 2nd pre.Re.)*

| 7 | 2 | 12 | 2 |
| --- | --- | --- | --- |
| 7 | 2 | 12 | 2 |

## Exercise 2

1. Reasons
   1. Sum of output values is bigger than sum of input values
   2. One of the used UTXOs is not owned by the TX-creating account
   3. No, accepting zero-confirmed TXs would lead to a security issue as this could possibly allow an attacker to perform double-spending patterns by using the money twice and hoping the second TX is mined first
   4. There are full and light nodes. Full nodes have a full copy of the blockchain and maintain it, perform validation, etc. Light nodes are connected to a full node and only have a view on the data. For our use-case a full-node would be good as we can directly monitor the mempool and are aware which TX have been placed into a block already.
2. This TXO was created by Alice, as Alice is the owner of the TXO/IN used in #2
3. Carol is allowed to spend this TXO
4. Balances
   1. Alice: 13
   2. Bob: 0
   3. Carol: 2
   4. Dave: 4
5. It is a bad idea because this would allow an attacker to use one PubKey that fits to the PubKeyHash in order to proof he can spend the values. Another PubKey (his own) can then be used to perform the signature check and therefore spend the funds himself.
   1. Attack-TXIn Script:
      1. attackerSig
      2. attackerPub
      3. pubhashOwnerPub
6. ASIC (at least the fastest), else FPGA (more energy efficient than GPUs)
7. 10 - 200 Exa-hashes
   1. Reason:
      1. Lower incentive for miners to mine blocks could lead to some of them deciding it’s not profitable anymore and therefore stop working. This again leads to less TX throughput.
   2. Resolved:
      1. By decreasing the difficulty
      2. By adding tips to base fees for TX, the incentive grew again (?) *not sure about that answer though*
      3. When the capacity of the mempool is full, it starts to add min tx fee and throw away txs that don't fulfil this criteria → tx fee would increase
8. Answers
   1. Downside
      1. Increasing the amount constantly would lead to inflation and worth loss of the currency, the scarcity is lowered and the miner incentive might be also reduced as a reason
   2. Alternative approach
      1. The fees of transactions might be required to be put up so miners do have an incentive again to process them. (base fee + tip)
9. Examples
   1. Soft-Fork: changing block size from 1 MB to 0.5 MB
   2. Hard-Fork: changing block size from 1 MB to 8 MB
   3. No impact: adding a new API endpoint
10. Answers
    1. Chain-Split
       1. A chain split happens if an update is voted and some part of the network decides in favor and some against it. If those voted in favor still want to update, they can perform a chain split meaning they now run independent
    2. Replay-Attack
       1. A replay attack can happen if a chainsplit occurred and someone had a UTXO before the split occurred. Now it is possible to use that UTXO in both new chains as they won’t ask each other anymore for approvals. Prevention can be to add a new genesis block in the chain or make it incompatible with old UTXOs.

## Exercise 3

1. Explanations
   1. Nonce: nonces are added to prevent replay attacks, as messages are not included in blocks nor mined it is not necessary there
   2. Gas Price: the gas price is inherited by the initial transaction triggering the messages + as the TX is already in execution (msgs are part of it) there is no reason to include it in the messages
2. Addresses
   1. 0xaaa: must be an externally owned account (EOA) as it has sent a TX and only EOAs can send TXs
   2. 0xbbb: must be a smart contract, as only smart contracts can issue messages
   3. 0xddd: we can not say anything about that as both EOAs and contracts can receive messages
3. Vars:
   1. msg.sender: 0xbbb
   2. tx.origin: 0xaaa
4. Vars:
   1. Sender: 0xAlice
   2. Recipient: 0xyyy
   3. Value: 0 ETH
   4. STARTGAS: 45000
   5. Gas used: 41000
   6. Gas Price: 70 Gwei
   7. Data: transfer(0xBob, 2) *not sure about the format*
   8. Advantage: account balance directly visible (?), could allow credits and deposits OR BETTER: useful for trading on on-chain decentralized exchanges
   9. Disadvantage: additional costs and complexity (OR) value might sometimes drop below 1 ETH since it is backed up by an algorithm or asset
5. 1. A: ERC721 - Non-Fungible-Tokens are the only logical token to use for unique things like real-estate, as they are also unique
   2. B: ERC20 - Fungible-Tokens are making more sense here, as the power is not unique and more like a currency which you can buy/have more than 1 of

## Exercise 4

Modifier

| modifier onlyOwner {  require(msg.sender == owner);  \_; } |
| --- |
| constructor () public  {  owner = payable(msg.sender); } |
| function setAuction … public onlyOwner … |
| function bid() public payable {  require(auctioned\_item != 0);  uint bid = uint(msg.value);  require(bid > auction\_item.price);   accountBalances[auctioned\_item.highest\_bidder] = accountBalances[auctioned\_item.highest\_bidder].add(auctioned\_item.price)  auctioned\_item.price = bid;  auctioned\_item.highest\_bidder = payable(msg.sender); } |
| function withdraw() public  {  unit balance = accountBalances[msg.sender];  require(balance > 0);  require(msg.sender != acutioned\_item.highest\_bidder);  accountBalances[msg.sender] = 0;  payable(msg.sender).transfer(balance); } |
| function finishAuction () public onlyOwner … |

## Exercise 5

1. Consensus is reached once the endorsement of a transaction is accepted, was ordered, the execution was valid and committed. Side notes:
   1. Approval works by committing peers that verify endorsements
   2. Endorsing peers in addition to verification, create endorsements for TXs by executing proposals
2. False ⇒ Execution-Order-Validate is used

True

False ⇒ there is no cryptocurrency in fabric

False ⇒ it was not deprecated

True

False ⇒ permissioned chain (MSP needs to allow you as a peer)

False ⇒ endorsing peer or committing peer (not validating)

True

True/False ⇒ chaincode can be written in go, nodejs and java (not native one)

True

## Exercise 6

Corda ⇒ not part of lecture

Exam 2021

## Exercise 1

1. It is not a hash function because it does not conform to the property of compression. This defines, that the values of h(x) must have the same bit length. An example proving that wrong is x1=1 and x2 =100 with h(x1)=18 and h(x2)=117, as they differ in decimal amount they also differ in bit length
2. It violates the property of collision. For x=1 and y=1025 g(x)=18 and g(y)=18 which means for x!=y still g(x)=g(y)
   1. It also violates 2nd preimage resistance…
3. After some rounds, the hash values for the choices will be known and then they are no secret anymore
4. We could add a random number to each choice, which will need to be revealed together with the selected choice, which adds a hiding property to the hash function
5. Exactly one, the merkle root hash
6. 

## Exercise 2

1. 1 the hash of the previous block header, 2 is the Nonce
2. By containing the hash of merkle root, all Transaction hashes are included as well
3. The coinbase transaction provides the mining reward for the miner, and it has the option to contain some data in the TXIN of it, that the miner can define. It also contains the block height.
4. If the time it takes to mine a block is getting too small, the difficulty is increased to keep a consistent mining speed

## Exercise 3

1. Each entity is identified by a public-key (unique key) and its hash
2. Bob has created it
3. Bob can spend it
4. Balances
   1. Alice: 13
   2. Bob: 0
   3. Carol: 2
   4. Dave: 3
5. The rest is the transaction fee
6. P2PKH, Pay-to-Public-Key-Hash, it makes the bitcoin address shorter + more secure (has it’s only the hash)
7. *Maybe P2SH*
8. scriptSig defining the signature of a TXIN and scriptPubKey defining the conditions of who can spend a UTXO.SegWit shortens the scriptSig.

## Exercise 4

1. Nakamoto / Proof of Work
2. Light and full nodes, only full nodes perform full validation. Light nodes only validate whether a single transaction of the wallet owner was executed
3. Reason: it’s hard to solve the puzzle alone, splitting the work increases the chance of winning. Disadvantage: mining pools centralize against the control mechanism which should be decentralized
4. Energy consumption
5. The block time will be reduced as it is now easier to mine a block
   1. Higher possibility of chain forking (leading to multiple realities
   2. Network has to keep track of these forks even if many will be orphaned
   3. Empty blocks
   4. Hash rate adapts to difficulty - goes down
   5. (Higher profitability of the mining operations? Assuming the same price and at least until the hashrate rises again) ⇒ Though this is no effect on the network
6. 51%
7. Taproot is a soft-fork
8. By using the version field, having 32 bits in each block reserved for voting on possible updates
9. No it will not. As it has support by at least 90% of the community (and therefore approx. also from the network more than the majority), it will not lead to a chain split. And works with the old blockchain. (Usually 95% are required, though this depends)

## Exercise 5

1. BTC uses a transaction based ledger. ETH instead, uses an account based ledger.
2. A nonce prevents so-called replay attacks. As ETH has an account based ledger, each transaction without a nonce would not have an ordering mechanism nor a duplication prevention mechanism. Using an increasing nonce, TX can not be duplicated as there can’t be two with the same nonce, nor can one TX be moved before another one by anyone else (as the nonce is strictly increasing).
3. As messages won't be included in the blockchain, there won't be any replay attacks. So a Nonce is not needed here. Messages are triggered by an origin transaction and created by a smart contract.
4. (sum of all gas used in one TX (including messages)) \* gas price
   1. 467,352\*70 = price in gwei
5. 0xaaa
6. There EOAs (externally owned accounts) and smart contracts. EOAs can only issue transactions (e.g., 0xaaa) whereas smart contracts can only issue messages (e.g., 0xbbb)
7. Values
   1. Sender: 0xAlice
   2. Recipient: 0xyyy
   3. Value: 0 ETH
   4. STARTGAS: 45000
   5. Gas used: 40000
   6. Gas Price: 30 Gwei
   7. Data: transfer(0xBob, 3)
8. A: ERC20 as there can be multiple of them and they are not unique (like shares)

B: ERC721: art is unique and therefore non fungible tokens fit best

1. Advantages:
   1. Less power consumption
   2. Predictable and faster finality
   3. Harder to attack since you need 2/3s majority
   4. Higher transaction throughput
   5. Can loose more as you loose stake (51% more unlikely)
   6. *More decentralization (mining pools not required anymore)*

## Exercise 6

| modifier onlyPlayer() {  require(msg.sender == owner || msg.sender == playerTwo);  \_; } |
| --- |
| constructor (address \_playerTwo) {  owner = payable(msg.sender); playerTwo = payable(\_playerTwo); …  } |
| function commit(bytes32 \_hashedChoice) public payable onlyPlayer {  require(msg.value >= 1 ether);  require(!isPlaced[msg.sender]);  if(isPlaced[owner] || isPlaced[playerTwo]) {  require(msg.value == bet);  } else {  bet = uint(msg.value);  }  isPlaced[msg.sender] = true;  commitments[msg.sender] = \_hashedChoice;  } |
| function reveal(uint \_choice, int \_nonce) public onlyPlayer {  require(isPlaced[owner]);  require(isPlaced[playerTwo]);  require(isRevealed[msg.sender] == false);  \_choice = \_choice % 3;  bytes32 claimHash = keccak256(abi.encodePacked(\_choice, \_nonce));  require(claimHash == commitments[msg.sender]);  isRevealed[msg.sender] = true;  revealedCommitments[msg.sender] = \_choice; } |
| function distributeWinnings() public onlyPlayer { require(isRevealed[owner] && isRevealed[playerTwo]); … } |

## Exercise 7

1. It is a permissioned blockchain where a Membership Service Provider governs access
2. A channel is an own blockchain which has member peers and its own chaincode (benefit) → isolation between chains (each having different purposes if required)
3. Types
   1. Committing Peer
      1. Validates endorsements and commit transactions
   2. Endorsing Peer
      1. Committing Peer + creates endorsements for transaction proposals
   3. Ordering
      1. Orders transactions into blocks
4. The committing peer, as it validates transactions/endorsements and maintains a state of the chain **OR** the ordering peer as it orders transactions and puts them into a block.

## Exercise 8

Corda ⇒ Not part of the lecture (will be Tezos most likely in the exam)