Here is a collection of exam problems taken from a recent blockchain course. Some of these problems are hard, some are easy. Some of them may require you to seek information not provided in lectures.

Problem 1. Which of the following does Nakamoto consensus guarantee (or guarantee with all but negligible probability)?

1. agreement
2. validity
3. termination

Solution:

1. agreement: no, there may be a fork
2. validity: yes, any decision is valid
3. termination: yes, with all but negligible probability

Problem 2. Consider a function

$$H : X \times Y \to \{0, \ldots, 2^n\},$$

where X is called the domain of *puzzles* and Y is called the domain of *solutions*.

We say that H is *collision-resistent* if it is hard to find $x, x' \in X$ and $y, y' \in Y$ such that $H(x, y) = H(x', y')$.

We say that H is *PoW-secure* if, for a fixed difficulty D, say $D = 2^{32}$, for any x it is hard to find a y such that $H(x, y) < 2^n/D$.

Show that a collision-resistant H may not be proof-of-work secure.

Hint: Let

$$G : X \times Y \to \{0, \ldots, 2^m\},$$

be a collision-resistant hash function, were m may be less than n. Construct from G a function H that is collision-resistent but not PoW-secure. Prove your claim.

Solution: Assuming $D = 2^{32}$, let $m = n - 32$. Define $H = G$. Clearly, H is collision-resistent, because so is G, but it is not PoW-secure, because $H(x, y) < 2^n/D = 2^{n-32} = 2^m$.

Problem 3. Alice can use a Merkle tree to commit to a set of elements $S = \{e_0, \ldots, e_{n-1}\}$ so that later she can prove to Bob that some e_i is in S using at most $\lceil \log n \rceil$ hash values.

Consider how to do the same with a 3-ary tree, where every leaf node has up to 3 children. The hash at every non-leaf node is the hash of the concatenation of the children.

1. Suppose $S = \{e_0, \ldots, e_8\}$. How does Alice prove to Bob that e_3 is in the tree?
2. If S contains n elements, what is the length of the proof that shows that some e_i is in S?
3. How long is the proof if we replace 3 with $k > 2$?

Solution:

1. Alice shows the parent hash, the three child hashes, all the way down to the target.
2. $\lfloor 2 \log n \rfloor$
3. $\lfloor (k-1) \log n \rfloor$

Problem 4. The notation $\{m\}_{\mathrm{PK}}$ denotes a pair $(m, \sigma$ where σ is a signature on m using the secret key PK.

The *DeadTreeCoin* cryptocurrency is the latest investment craze. Transactions are published in newspaper classified ads, and they look like this:

$$\begin{aligned}
\mathrm{coin}_1 &:= \{\text{create 1 coin serial 1234 for } \mathrm{PK}_{\mathrm{fed}}\}_{\mathrm{PK}_{\mathrm{fed}}} \\
\mathrm{coin}_2 &:= \{\text{pay SHA256}(\mathrm{coin}_1) \text{ to } \mathrm{PK}_{\mathrm{mph}}\}_{\mathrm{PK}_{\mathrm{fed}}} \\
\mathrm{coin}_3 &:= \{\text{pay SHA256}(\mathrm{coin}_2) \text{ to } \mathrm{PK}_{xyz}\}_{\mathrm{PK}_{\mathrm{mph}}}
\end{aligned}$$

Every day the paper publishes a classified ad that looks like:

> new-coins, SHA256(yesterday's ad).

Here is a typical transaction.

$$\mathrm{coin}_i := \{\text{pay 0xcafebabe to } \mathrm{PK}_{foo}\}_{\mathrm{PK}_{bar}}$$

Define rules for deciding whether this transaction is valid.

Solution: The rule is inductive: a coin is valid when it is minted. When a valid coin is spent by the coin's owner, the old coin becomes invalid and the new coin becomes valid. A coin is valid if there is a chain of such spends following from a minted coin.

Problem 5. Suppose there are two distinct Bitcoin protocol implementations, called A and B. One day an attacker finds a vulnerability in implementation A that causes miners running that implementation to accept transactions that double-spend a UTXO. Implementation B treats such transactions as invalid.

- Suppose 80% of the mining power runs A and 20% runs B. What will happen to the blockchain once a block containing a double-spending transaction is submitted to the network?
- Suppose 80% of the mining power runs B and 20% runs A. What will happen to the blockchain once a block containing a double-spending transaction is submitted to the network?

Solution:

- If 80% of the mining power runs A, then A miners will drive out B miners. The miners running A will build long chains not recognized by B, and the miners running B will lag behind.
- If 80% of the mining power runs B, then double spending blocks will die out. The miners running B will ignore the double spending blocks, and they will outcompute the miners in A.

Problem 6. Consider the following Solidity function contained in an Ethereum contract.

```
1  function wasted(uint246 x, uint256 y) {
2    while (uint256(sha3(x))) != y) { // sha3 is a hash function
3      x++;
4    }
5  }
```

Assume you know an x and a y for which this function never terminates. Explain how a malicious miner can use this function to waste other miners' resources without spending any of its own resources. *Solution:* The malicious miner could put the contract in a block, along with a function call using the non-terminating argument values, but not execute that code (spending no resources). Every other miner attempting to validate that block will try to execute that function, wasting resources until it decides to quit.

Problem 7. *Rock, Paper, Scissors* (RPS) is a two-player game where each player simultaneously chooses one of three values: *Rock, Paper, Scissors*. The winner is determined by the following rules: Scissors beats Paper, Paper beats Rock, and Rock beats Scissors.

Your mission is to write a contract allowing Alice and Bob to play RPS on the blockchain. Worry about these two possibilities:

1. one player may try to observe her opponent's move before choosing her own, and
2. one player may walk away from the game midway, or just take too long.

Describe, either in pseudo-code or in simple English, how you would design a RPS contract to address these possibilities. (We are interested in big ideas here, not code details.)

Solution:

1. Use the commit-reveal pattern to prevent one player for seeing the other's move.
2. If one player fails to reveal within a fixed time after the other, the first one wins.

Problem 8. Figure **??** shows a simple bank where a user can deposit funds with a *timelock*, establishing a duration during which the funds cannot be extracted. This contract is useful to *HODL* enthusiasts when currencies are volatile and faith begins to waver. Is there any way a panicky user can extract their funds early? Explain why or why not.

Solution: Yes, call the increaseLockTime() with an argument that causes the user's lockTime to overflow to zero.

Problem 9. Sports betting is a big business, so imagine we have a contract allowing clients to bet on the outcome of a sports event. Each Client records its addresses, the amount of its bet, and the team it thinks will win. Betting is halted before the game begins.

Problem is, blockchains don't watch cable TV. How does the sports betting contract know who won the game? There may be a lot of money at stake, so losing clients are motivated to fool the contract into deciding that the wrong team won.

Here is one solution. Before the contract is published, we pick three *witnesses* who are hired to tell the contract who won the game. These witnesses do not know each other, and have no way to communicate outside their pubic blockchain votes. Each witness is paid handsomely to vote, but only if it votes with the majority of the witnesses. Witnesses may be *corrupt*, bribed to vote for a particular team no matter who won the game. No witness knows whether any other witnesses is corrupt. Witnesses who are not corrupt will, however, lie if they can profit from lying.

How should you structure the contract to ensure that any uncorrupted witness has an incentive to vote for the actual winner? Describe the contract either in pseudo-code, or in simple English. As always, we are interested in the big issues, not programming details.

Solution: The contract must use the commit-reveal pattern because otherwise a corrupt witness could vote first for the losing team, and an uncorrupted witness who sees that vote will vote the same, because then it is sure of getting paid.

Problem 10. Alice and Bob want to trade 1 of Alice's redcoins for 1 of Bob's greencoins. Suppose time Δ is long enough for either Alice or Bob to change a blockchain's state and for the other party to notice.

Alice proposes the following protocol to Bob. Alice will create a random secret s, where $h = H(s)$, for $H(\cdot)$ a well-known cryptographic hash function. She will create contact C_A, initialized with one redcoin, h, and Δ:

- If Bob presents s such that $h = H(s)$ before time 2Δ elapses, then transfer the coin to Bob.
- Otherwise time out and refund the coin to Alice.

Bob will create contract C_B, initialized with one greencoin, h, and Δ:

- If Alice presents s such that $h = H(s)$ before time 3Δ elapses, then transfer the coin to Alice.
- Otherwise time out and refund the coin to Bob.

They will follow this protocol:

1. At time 0, Alice publishes C_A on the redcoin blockchain.
2. When Bob sees that Alice has published C_A, he publishes C_B on the greencoin blockchain.
3. When Alice sees that Bob has published C_B, she sends s to C_B, transferring the greencoin to her.
4. When Bob sees that Alice has revealed s by sending it to C_B, he sends s to C_A, transferring the redcoin to him.

Should Bob accept? (Either explain why the protocol works, or find a way one side can cheat the other). *Solution:* Alice can cheat Bob. Assume everyone runs as slowly as possible.

time	step
0	Alice publishes C_A
Δ	Bob notices C_A, publishes C_B
$2\Delta - \epsilon$	Alice sends s to C_B, greencoin transferred to Alice
2Δ	C_A times out, transfers redcoin to Alice

At this point, Bob has lost his coin, but cannot get Alice's coin.

Problem 11. Consider the following proposed Byzantine agreement protocol. As usual, there are $3f + 1$ validators, where at most f can be dishonest. One validator can send messages directly to another, but these messages cannot be observed by a third validator. Messages are always delivered on time, and they are signed, so no validator can forge another's signature. One validator is designated leader.

1. the leader picks a block b.
2. The leader sends a *propose* message containing b to every validator, including itself.
3. When a validator gets a propose message for b, it sends a *vote* message for b to every validator, including itself.
4. If a validator receives at least $2f + 1$ votes for b, it *decides* b, and if it does not get $2f + 1$ votes for any single block, it *decides* $\perp$.

True or false? Give a one-or-two sentence justification.

1. If an honest leader proposes b, then every honest validator decides b.
2. A dishonest leader can trick some validators into deciding b and others into deciding b', where neither b nor b' is $\perp$.
3. One honest validator might decide a block $b \neq \perp$, while another honest validator might decide $\perp$.

Solution:

1. If an honest leader proposes b, then every honest validator decides b. *True:* $2f + 1$ *honest validators all receive* b*, they send to one another, and all decide* b*.*
2. A dishonest leader can trick some validators into deciding b and others into deciding $b' \neq b$, where neither b nor b' is $\perp$. *False: whoever decided* b *observed at least* $2f + 1$ *votes for* b*, and whoever decided* b' *observed at least* $2f + 1$ *votes for* b'*. These two sets intersect in at least* $f + 1$ *validators, implying that at least one honest validator lied about its vote.*
3. If one honest validator decides b, they all do, and if one honest validator decides $\perp$, they all do. *False: a dishonest leader can propose* b *to* $f + 1$ *honest validators,* b' *to* f *honest validators, and* f *dishonest validators can cause the* $f + 1$ *to decide* b *and the rest to decide* $\perp$*.*

Problem 12. Recall that, legally, a *custodian* is trusted financial adviser in a position to steal or abuse a client's assets. Two advisers are *operationally independent* if they can be reasonably expected to act independently. For example, they would *not* be operationally independent if they both work for the same organization. Suppose Xavier and Yves can steal an asset, but only by working together. If they are operationally independent, they are not custodians, and otherwise they are.

The Orinoco[1] corporation is a major seller of on-line merchandise. They decide to get into the blockchain purchase mediation business as follows. When Alice wants to buy something from Bob, he ships her the goods, and she puts her money is a 2-out-of-3 signature escrow contract, where Alice owns one key, Bob the other, and the Orinoco service owns the third.

If Alice is happy with the goods, she and Bob both sign and transfer the money to Bob. If Alice is unhappy, they appeal to the Orinoco corporation service for mediation. If the service finds for Alice, Alice and Orinoco sign to refund the money to Alice, otherwise Bob and Orinoco sign to award the money to Bob.

You are Orinoco corporation's lawyer, in charge of advising the company whether it is in danger of being regulated as custodian. Please give a one or two-sentence defense of *one* of these positions.

1. No, Orinoco corp would not be legally considered a custodian for the following reasons ...
2. No, Orinoco corp would be legally considered a custodian for the following reasons ...
3. It depends, Orinoco corp would be considered a custodian under the following circumstances ...

Solution: If either Alice or Bob was affiliated with Orinoco, then the corporation is in a position to steal or abuse the escrowed money.

[1] The second biggest river in South America

Problem 13. Consider the privacy implications of a Bitcoin-like cryptocurrency that has two kinds of UTXO. The first kind of UTXO has format:

0x21	in0	in1	out	(other stuff ...)

A UTXO with header *0x21* has two input addresses, occupying slots 2 and 3, and one output address, occupying slot 4. The remaining fields include signatures, hashes, and other fields needed to make the blockchain work. This UTXO assigns the balances of both in0 and in1 to out.

The second kind of UTXO has the format

0X12	in	out0	out1	(other stuff)

A UTXO with header *0x12* has one input address, occupying slot 2 and two output addresses, occupying slots 3 and 4. This UTXO splits the balance of in evenly between out0 and out1.

The Josiah Carberry private wallet runs on this blockchain. The wallet keeps all the user's currency in one address, denoted balance. Suppose Alice wants to transfer money into Bob's wallet, The wallet software generates an address temp to which Alice transfers her money, along with a random address newBalance, never used before, to hold the wallet's new balance. The wallet software merges Alice's contribution with the wallet's balance by placing this UTXO on the blockchain:

0x21	balance	temp	newBalance	(other stuff ...)

Here balance in slot 2 is the address holding the wallet's current balance, and temp and newBalance in slots 3 and 4 are the addresses described above.

temp and balance are never used again.

Suppose Bob wants to give Carol half his money. He gives the wallet an address carol from Carol, and the wallet software generates a random address newBalance, never used before, to hold the wallet's new balance. The wallet software places this UTXO on the blockchain:

0x12	balance	carol	newBalance	(other stuff ...)

Here balance in slot 2 is the address holding the wallet's current balance, and carol and newBalance in slots 3 and 4 are the addresses described above.

The Carberry wallet company claims their wallet is highly private.

- Someone monitoring the blockchain cannot follow the wallet balance over time because the balance address holding the wallet's current balance changes unpredictably with each transaction.
- There is no attack that can link a wallet to a user because the wallet software never leaks the current balance address.

Explain in one or two sentences:

- Is the first claim accurate? Does the blockchain format leak information about the wallet's current balance?
- Is the second claim true? Is there an attack covered in class that can link this wallet with a user?

Solution:

- Someone monitoring the blockchain cannot follow the wallet balance over time because the balance address holding the wallet's current balance changes unpredictably with each transaction.

  Wrong. For each UTXO, the wallet's old balance address is always in slot 2, and the new balance address is in slot 4, so it is easy to track any wallet's balance.

- There is no way to link a wallet to a user because the wallet software never leaks the current balance address.

  Wrong. A “dusting” attack as discussed in class can easily link a user with a wallet.