`1a) Every cryptographic hash function is a hash function. But not every hash function is a cryptographic hash.

A cryptographic hash function aims to guarantee a number of security properties. Most importantly that it's hard to find collisions or pre-images and that the output appears random. (There are a few more properties, and "hard" has well defined bounds in this context, but that's not important here.)

Non cryptographic hash functions just try to avoid collisions for non-malicious input. Some aim to detect accidental changes in data (CRCs), others try to put objects into different buckets in a hash table with as few collisions as possible.

Bi,ii,ii) in Slides

c) in slides

d) Pre-Image Resistant: For a given h in the output space of the hash function, it is hard to find any message x with H(x)=h.

We can create a simple Hash function H(x) = x. It is difficult to find a pair of x and y that has, same hash value, but it is extremely easy to identify the x. (? Really Not Sure if this is Okay)

Consider a hash function g: {0, 1}\* → {0, 1}n. We assume that g is collision resistant. We now construct h: {0, 1}\* → {0, 1}n+1 of n + 1 output bits from g as follows.

If the length of x is n, h(x) is 0 || x. Otherwise, h(x) is 1||g(x).

Observe that h is collision resistant, since 1 concatenated with g is collision resistant, and 0 concatenated with the input is also collision resistant. However, h is not preimage resistant, as all hashes beginning with 0 are trivially invertible. (Would this work as well?)

e) It is easier to break its resistance to collision than to preimage. For successfully 'breaking' preimage resistance, you get some output hash x and need to find the corresponding input m to that specific x. Unless there is a structural flaw in the hash algorithm, the quasi-randomising effect of the hash function dictates that the only way is to brute-force over a lot of different inputs until you find your target input m. This is extremely expensive, and intentionally so.

For collisions, you need to find any two distinct inputs m1,m2 that result in the same output hash. Now again you could choose a brute-force approach and try different inputs until you find a match. However you are not constrained by the fact that your inputs must result in a specific hash, like in both preimage cases. This makes it significantly easier to 'break' collision resistance.

f) i) 8.53 Hashrate ii) 64 mins

2ai) Proof of work makes it hard for people to change previous transactions/blocks because of the difficulty involved in reaching a target. This prevents people from double spending and fraud, the longest chain is the chain that has the most proof of work and records the transaction. This acts as a way for people to reach consensus. The main chain is defined as the longest chain with the most proof of work

2a ii) We may also need a broadcast model to achieve the necessary agreement on a single data value or a single state of the network among distributed processes ? (I would add that we need the full history of transactions and a set of consensus rules. We can’t use proof of work without having all the blocks as we don’t know that the block, we are building on top of is valid or not, and we also need a set of rules to distinguish what is valid and what is not.)

b) If an entity has 50 % of hashing power it could flout the rules temporarily, allowing double-spending of coins and blocking transactions, thus people might not be able to reach consensus.

b)Not really sure what it means. A consistent broadcast means getting information of txn/blocks every certain period of time( every 10 mins), this ensures that you do not fall behind in the number of blocks and is always updated. A reliable broadcast on the other hand, means that the block/txn being advertised is not made by an adversary, the txn/block broadcasted is not a doublespend txn of a fork of the main chain.

Other answer:

* **Consistent** broadcast: A protocol for consistent broadcast satisfies:

**Validity**: If a correct sender *Ps* broadcasts m, then all correct parties eventually deliver m.

**Consistency**: If a correct party delivers m and another correct party delivers m' , then m = m'

**Integrity**: Every correct party delivers at most one request. Moreover, if the sender Ps is correct, then the request was previously broadcast by *Ps*

* **Reliability**: Consistent + Totality (If some correct party r-delivers a request, then all correct parties eventually r-deliver a request)

c)i)Consistency in this problem means each country will have the same attack time proposal and will advertise the same time and will attack at the same time. Reliability means each country will not intentionally give the wrong info to another party.

ii) PoW to store the attack times in a blockchain ?

d) The problem of consensus - that is, getting a distributed network of processors to agree on a common value - was known to be solvable in a synchronous setting, where processes could proceed in simultaneous steps. In particular, the synchronous solution was resilient to faults, where processors crash and take no further part in the computation. Informally, synchronous models allow failures to be detected by waiting one entire step length for a reply from a processor, and presuming that it has crashed if no reply is received.

This kind of failure detection is impossible in an asynchronous setting, where there are no bounds on the amount of time a processor might take to complete its work and then respond with a message. Therefore it’s not possible to say whether a processor has crashed or is simply taking a long time to respond. The FLP result shows that in an asynchronous setting, where only one processor might crash, there is no distributed algorithm that solves the consensus problem.

e) Block reward and transaction fees incentivise honest node behaviour by providing consequences and rewards. A miner is incentivised to not be malicious in the hope they receive a block reward (part of BTC) and therefore gain something for mining, if that block is approved by consensus. These transaction fees make up the reward to miners, and the higher the reward helps the miners prioritise which blocks to commit to the chain first.

f)

i) orphan blocks occur when 2 different miners generate a block simultaneously. The rejected one are the one which have relatively lower share of Pow and is known as the orphan block, orphan blocks results in sunk computational costs. The name is misleading because orphaned blocks are not actually orphans because they do have parents in the form of the previous blocks in the chain, a stale block is probably a more correct terminology.

ii) Stale block rate is affected by two thing, block generation rate and blocksize. A higher block generation rate means a higher stale block rate since block are generated quicker, a larger blocksize means blocks are propagated slower, people receive correct block info slower more stale blocks. I don’t know the mathematical expression but I am guessing R=alpha(Generation)+beta(Size). Stale block rate proportional to both size and block generation.

3

i)Storage very expensive, the contract storage values are stored in the blockchain forever

Memory- cheaper than storage, only accessible during contract executive, once execution is finished contents are discarder

Stack- The compiler generally uses it for intermediate values in computations and other scratch quantities.

ii) Fallback function is an unnamed function in solidity it has no arguments and doesn’t return anything. This f(x) is executed when if none of the other functions match the given function identifier or no data supplied at all.

iii) No, because Ethereum computers are usually slower than real life computers. Therefore, a miner can actively choose to invent a future (i.e., mine a block) whose “random” properties will yield a favourable outcome.

Bi)msg.sender refers to the account or contract that invokes the function, tx.origin refers to the original external account that started the transaction chain

Contract chain A -> B -> C (msg.sender in C will return B whereas tx.origin will return A)

ii) Not really sure, require that u don’t have the same address as tx.origin, x value is 0 and ur \_gatekey is –1 (I think \_gatekey should be 2^64 – 1 as the numbers are uint so they would underflow)

Ans:

iii) yes the transaction is recorded in the blockchain and cannot be altered s

4a) Transaction processing capacity of bitcoin network is 10 minutes per block and blocksize limit of 1 MB. Number of transactions in a block cannot be more than 1MB in size else it would be pushed to next block and have to wait 10 minutes more.

Bi)More demand of blockspace, txn fee will increase, lower the demand lower the txn fees

ii) Larger transaction fee => faster to be considered a successful and valid transfer. Lower transaction fee => Need to wait more time / not be included in the blockchain

c) Larger blocks make full nodes more expensive to operate. Therefore, larger blocks lead to less hashers running full nodes, which leads to centralized entities having more power

d) In slides

e)

Setup:

1. Alice and Bob each sends a sum of money they want to commit to the multisig wallet smart contract in the setup transaction. The smart contract keeps track of how much money each person has sent.

Spending and refund:

1. Alice and Bob must each send a tx with their signatures to approve the spending of funds. Once everyone has approved, then the smart contract will forward the call on to transfer the funds, or call another smart contract.

2. At the same time, the smart contract keeps track of the timing where each person approved. It then starts a timelock for the other parties to approve the transfer.

3. If not all parties approve by a certain amount of time, the multisig wallet is “unlocked”, and Alice is able to call a refund function to withdraw her own funds.

f)

i) It is called an invalidation tree, because the siblings of the tree with a lower timelock T will invalidate the states of the other siblings. This is called time lock state replacement. It works by having transactions only being able to be broadcast after its timelock has been passed. Therefore, siblings with lower timelocks will invalidate the others as before, as they can be included in the blockchain at an earlier block number, and double spending prevents the other siblings from being broadcast.

ii) In a simple timelocked payment channel, one quickly runs out of time by doing transactions in the channel, each requiring a smaller timelock on the commitment transaction. This was solved with a tree of transactions as shown in Figure 2. At any point in time, only the path where all transactions have the lowest timelock of their siblings can be broadcast. In this way, many commitment transactions can be created before the timelocks get too low and the channel cannot be updated any more.