(2½ Hours)

[Total Marks: 60]

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labelled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is**allowed**.

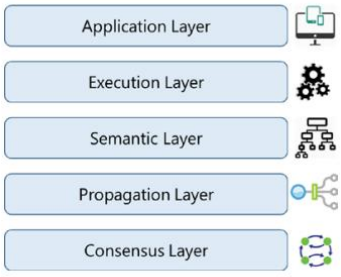
I	**Choose the correct alternative and rewrite the entire sentence with the correct alternative.(30)**			
1.	_____ is a peer-to-peer system of transacting values with no trusted third parties in between and have a shared, decentralized, and open ledger of transactions.			
	a.	**Blockchain**	**b.**	Bitcoin
	c.	TCP/IP	**d.**	WWW
2.	In which system there is no "master" node as such and yet the computation may be distributed?			
	a.	Centralized system	**b.**	**Distributed system**
	c.	Master system	**d.**	Client system
3.	Identify the correct property: "an entity (a person or a system) cannot refuse the ownership of a previous commitment or an action ".			
	a.	Confidentiality	**b.**	Data Integrity
	c.	Authentication	**d.**	**Non-repudiation**
4.	In Advanced Encryption Standard (AES), the number of encryption rounds depends on _____.			
	a.	**The key length**	**b.**	The data block
	c.	The data size	**d.**	The key type
5.	The very first block is called as ____ as it does not contain any reference to the previous blocks.			
	a.	**The genesis block**	**b.**	The first block
	c.	The genuine block	**d.**	The complete block
6.	_____ is actually a list of instructions recorded with each transaction that describes how the next person can gain access to those Bitcoins received and spend them.			
	a.	**A script**	**b.**	A block
	c.	A signature	**d.**	A command
7.	In Etherum _____ is used for security during transaction.			
	a.	Symmetric cryptography	**b.**	Asymetric Crptography
	c.	Hashing techniques	**d.**	**Hashing and Asymetric Cryptography**

8.	EVM stands for___.			
	a.	Electric Virtual Machine	**b.**	Etherum Vision Mashine
	c.	**Etherum Virtual Machine**	**d.**	Electric Vision Machine
9.	____ is not a tool to work with Etherum in Mist.			
	a.	MetaMask	**b.**	Gest
	c.	**MistMask**	**d.**	Parity
10.	In Etherum 'smart contract' is___			
	a.	Node rules	**b.**	**Business rules implied by the contract in blockchain**
	c.	legal rules written in English	**d.**	Money rules
11.	___ is the file extension for solidity.			
	a.	.sl	**b.**	**.sol**
	c.	.solidity	**d.**	.s
12.	Solidity is ___type of programming language.			
	a.	Procedure Oriented	**b.**	**Object Oriented**
	c.	Scripting Language	**d.**	Low level language
13.	Gas amount is specified in			
	a.	**Wei**	**b.**	Ether
	c.	Bitcoin	**d.**	Satoshi
14.	What are contracts called in Hyperledger?			
	a.	Smart Contracts	**b.**	**Chaincode**
	c.	Fabric	**d.**	Ledger
15.	How many frameworks does Hyper Ledger have?			
	a.	4	**b.**	**6**
	c.	5	**d.**	9
16.	A ____ instrument used to transfer value between two parties over a blockchain network.			
	a.	Dapp	**b.**	**token**
	c.	key	**d.**	value
17.	Network Connection Profile is stored as _____ file.			
	a.	XML	**b.**	**JSON**
	c.	dba	**d.**	cto
18.	Task of miners in blockchain is____.			
	a.	Acts as a single third party.	**b.**	Ensures number of corrupt nodes will stay low.
	c.	Are responsible to access blockchain.	**d.**	**Compete for a reward by trying to**

				calculate the nonce.
19.	In Ethereum an Uncle refers to _____.			
	a.	Spy agent	**b.**	Protocol
	c.	**A non main block**	**d.**	A crowd funding project
20.	Name the Currency that exist in a distributed decentralized network?			
	a.	Sidechain	**b.**	Clone
	c.	**Ether**	**d.**	Exchange
21.	The Term Fork is used when _____.			
	a.	Blockchain joins	**b.**	Blockchain merges
	c.	Blockchain slices	**d.**	**Blockchain splits**
22.	Proof of Stake refers to ?			
	a.	Certificate	**b.**	Protocol
	c.	Key	**d.**	**Consensus**
23.	What is present in the top most node of a Ethereum Merkle tree?			
	a.	**Root hash**	**b.**	Genesis block
	c.	Child hash	**d.**	Nonce
24.	ECDS Algorithm stands for?			
	a.	Energy curve digital signature	**b.**	**Elliptic curve digital signature**
	c.	Encryption curve digital signature	**d.**	Ether curve digital signature
25.	A Bitcoin transaction is mostly just a transfer of_____ from one address to another.			
	a.	**Bitcoins**	**b.**	Information
	c.	Bits	**d.**	Data
26.	To execute a function of the smart contract, first we need to create _____ class with the ABI and address of our deployed contract.			
	a.	a instance of the web3.bitcoin.Contract	**b.**	a instance of the web3.Contract
	c.	**a instance of the web3.eth.Contract**	**d.**	a instance of the eth.Contract
27.	What is UTXO?			
	a.	United Transaction Office	**b.**	Union of Texas Operations
	c.	United Texan Xerox Organization	**d.**	**Unspent Transaction Output**
28.	geth stands for _____.			
	a.	Go-header	**b.**	**Go-Ethereum**
	c.	get-header	**d.**	get- Ethereum
29.	_____ is the value that limits the maximum amount of gas the code execution can consume when triggered by the message.			
	a.	gasEnd	**b.**	gasFinished

	c.	**gasLimit**	**d.**	gasExpire
30.	_____ is a set of accounts if any, that will be discarded after the transaction completion			
	a.	**Self-destruct set**	**b.**	Self-destroy set
	c.	Log Series set	**d.**	Refund Balance set

II	**Attempt *any one* of the following:**		**6**
	a)	Explain with diagram the various layers of blockchain.	



Various layers of blockchain

Application Layer :

This is the layer where you code up the desired functionalities and make an application out of it for the end users. It usually involves a traditional tech stack for software development such as client-side programming constructs, scripting, APIs, development frameworks, etc. For the applications that treat blockchain as a backend, those applications might need to be hosted on some web servers and that might require web application development, server-side programming, and APIs, etc.

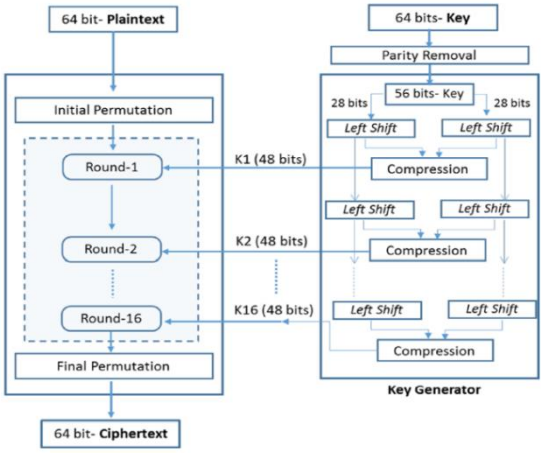
Execution Layer

The Execution Layer is where the executions of instructions ordered by the Application Layer take place on all the nodes in a blockchain network. The instructions could be simple instructions or a set of multiple instructions in the form of a smart contract. In either case, a program or a script needs to be executed to ensure the correct execution of the transaction. All the nodes in a blockchain network have to execute the programs/scripts independently. Deterministic execution of programs/scripts on the same set of inputs and conditions always produces the same output on all the nodes, which helps avoid inconsistencies.

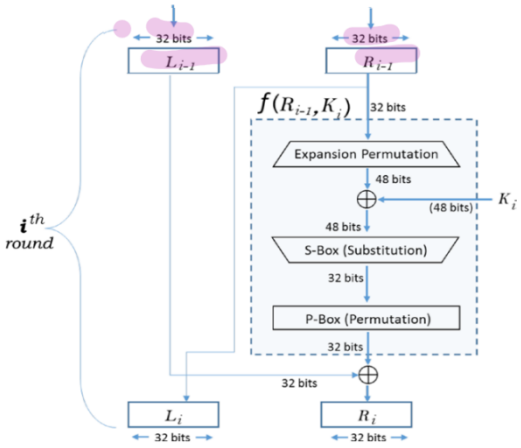
Semantic Layer

The Semantic Layer is a logical layer because there is orderliness in the transactions and blocks. A transaction, whether valid or invalid, has a set of instructions that gets through the Execution Layer but gets validated in the Semantic Layer. In this layer, the rules of the system can be defined, such as data models and structures. There could be situations that are a little more complex compared with simple transactions. Complex instruction sets are often coded into smart contracts. The system's state gets updated when a smart contract is invoked upon receiving a transaction.

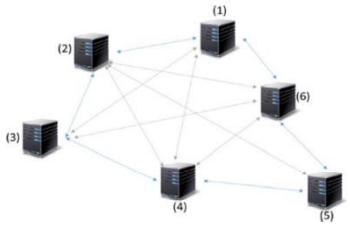
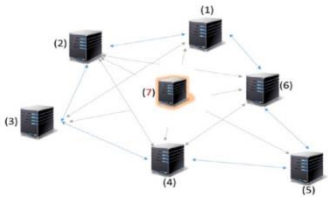
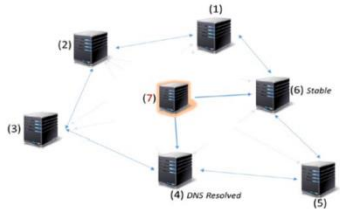
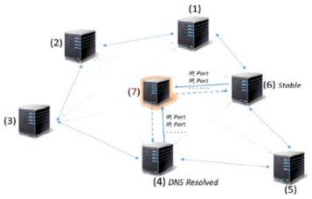
Propagation Layer

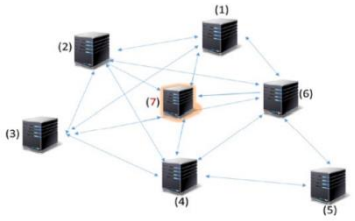
		The Propagation Layer is the peer-to-peer communication layer that allows the nodes to discover each other, and talk and sync with each other with respect to the current state of the network. Transaction / block propagation in the network is defined in this layer, which ensures stability of the whole network. By design, most of the blockchains are designed such that they forward a transaction/block immediately to all the nodes they are directly connected to, when they get to know of a new transaction/block. **Consensus Layer** The Consensus Layer is usually the base layer for most of the blockchain systems. The primary purpose of this layer is to get all the nodes to agree on one consistent state of the ledger. There could be different ways of achieving consensus among the nodes, depending on the use case. Safety and security of the blockchain is accertained in this layer. In Bitcoin or Ethereum, the consensus is achieved through proper incentive techniques called "mining."	
	b)	Explain Data Encryption Standard (DES) cryptography mechanism. Data Encryption Standard The Data Encryption Standard (DES) is a symmetric block cipher technique. It uses 64-bit block size with a 64-bit key for encryption and decryption. Out of the 64-bit key, 8 bits are reserved for parity checks and technically 56 bits is the key length. In symmetric cryptography, a large number of block ciphers use a design scheme known as a "Feistel cipher" or "Feistel network." A Feistel cipher consists of multiple rounds to process the plaintext with the key, and every round consists of a substitution step followed by a permutation step. The more the number of rounds, the more secure it could be but encryption/ decryption gets slower. The DES is based on a Feistel cipher with 16 rounds. A general sequence of steps in the DES algorithm is shown in Figure  DES uses the Feistel cipher rounds for encryption: • First, the plaintext input is divided into 64 bit blocks. If the number of bits in the message is not evenly divisible by 64, then the last block is padded to make it a 64-bit block. • Every 64-bit input data block goes through an initial permutation (IP) round. It simply permutes, i.e., rearranges all the 64-bit inputs in a specific pattern by transposing the input blocks.	

- After the IP round, the 64-bit block gets divided into two 32-bit blocks, a left block (L) and a right block (R). In every round, the blocks are represented as Li and Ri, where the subscript "I" denotes the round. So, the outcomes of IP round are denoted as L0 and R0.
- Now the Feistel rounds start. The first round takes L0 and R0 as input and follows the following steps:
- The right side 32-bit block (R) comes as is to the left side and the left side 32-bit block (L) goes through an operation with the key k of that round and the right side 32-bit block (R) as shown following:
- Li = Ri −1
- Ri = Li −1 ⊕ F(Ri −1, Ki) where "I" is the round number
- The F() is called the "Cipher Function" that is actually the core part of every round. There are multiple steps or operations that are bundled together in this F() operation.
- In the first step, operation of the 32-bit R-block is expanded and permuted to output a 48-bit block.
- In the second step, this 48-bit block is then XORed with the 48-bit subkey supplied by the key generator of the same round.
- In the third step, this 48-bit XORed output is fed to the substitution box to reduce the bits back to 32 bits. The substitution operation in this S-box is the only nonlinear operation in DES and contributes significantly to the security of this algorithm.
- In the fourth step, the 32-bit output of the S-box is fed to the permutation box (P-box), which is just a permutation operation that outputs a 32-bit block, which is actually the final output of F() cipher function. • The output of F() is then XORed with the 32-bit L-block, which is input to this round. This XORed output then becomes the final R-block output of this round.
- The output of F() is then XORed with the 32-bit L-block, which is input to this round. This XORed output then becomes the final R-block output of this round.



Once all the 16 rounds are over, the output of the 16th round is again swapped such that the left becomes the right block and vice versa. Then the two blocks are clubbed to make a 64-bit block and passed through a permutation operation, which is the inverse of the initial permutation function and that results in the 64-bit ciphertext output.

	c)	Explain with diagram - "How a new node becomes a part of the network". **Step-1:** Imagine that there were six nodes active at some point in time in the Bitcoin network.  Figure : Bitcoin network in general **Step-2:** There is a new node, say, a seventh node that just showed up and is trying to join the existing Bitcoin network, but does not have any connection yet.  Figure : A new node trying to join the network **Step-3:** The seventh node will try to reach out to as many nodes as it can either using DNS seeds or using the list of stable Bitcoin nodes in the list that it has  Figure: New Bitcoin node contacts some peers To connect to a new peer, the node establishes a TCP connection on port 8333 (could be different). Then the two nodes handshake with information such as version number, time, IP addresses, height of blockchain, etc. **Step-4:** In the fourth step, the requested nodes will respond with the list of IP addresses and corresponding port numbers of the other active Bitcoin nodes that they are aware of. The port number is important because once the TCP packets reach the destination node; it is the port number that is used by the operating system to direct the message to the correct application/process running on the system.  Figure : Peer Bitcoin nodes respond to the network request by a new node	

		Note that, only one peer may be enough to bootstrap the connection of a node to the Bitcoin network; the node must continue to discover and connect to new peers. This is because nodes come and go at will and no connection is reliable. **Step-5:** In the fifth step, the new seventh node establishes connection with all the reachable Bitcoin nodes, depending on the list it received from the nodes contacted in the previous step.  Figure: A new node becomes a part of the Bitcoin network	
2	**Attempt *<u>any one</u>* of the following:**		**6**
	a)	Write a short note on "Sending and Receiving Ether". Sending ether requires first holding some ether. On the main network, tokens either cost money or can be mined. However, this is an unwieldy way to get started for most Ethereum beginners. We've created an account on the main network, just in case you're interested in holding real ether for speculative value, or if you already have friends and collegues who use it for payments. For most readers, using test ether (which you can generate for free on the testnet, dubbed Ropsten) is better than paying money for real ether for use on the main network. Ether is sent via the Send dialog box. To send ether, you follow these steps: 1. In real life, ask the recipient for their Ethereum address. 2. Open Mist. Click Send in the top bar of the Mist wallet. The Send dialog box opens. 3. Choose which wallet you would like to send from, 4. Paste in the recipient's address. 5. Enter the amount. 6. Click Send. You'll notice two more options that you can toggle: a data field for entering extra text (for example, an order number or thank-you note) and a slider bar for choosing a transaction fee. To receive ether, your node does not have to be synchronized. If you'd like to check your balance, you can safely click Launch Application and skip the synchronization process when Mist launches.	
	b)	Define six steps of the Ethereum state transition function for each transaction in a block, the EVM performs. The Ethereum state transition function can be defined as the following six steps. For each transaction in a block, the EVM performs the following:	

		1. Check whether the transaction is in the right format. Does it have the right number of values? Is the signature valid? Does the nonce—a transaction counter—on the transaction match the nonce on the account? If any of these are missing, return an error. 2. Calculate the transaction fee by multiplying the amount of work required (represented by STARTGAS) by the gas price. Then deduct the fee from the user's account balance, and increment the sender's nonce (transaction counter). If there's not enough ether in the account, return an error. 3. Initialize the gas payment; from this point forward, take off a certain amount of gas per byte processed in the transaction. 4. Transfer the value of the transaction—the amount being sent—to the receiving account. If the receiving account doesn't exist yet, it will be created. (Offline Ethereum nodes can generate addresses, so the network may not hear of a given address until a transaction takes place.) If the receiving address is a contract address, run the contract's code. This continues either until the code finishes executing or the gas payment runs out. 5. If the sending account doesn't have enough ether to complete the transaction, or the gas runs out, all changes from this transaction are rolled back. A caveat is the fees, which still go to the miner and are not refunded. 6. If the transaction throws an error for any other reason, refund the gas to the sender and send any fees associated with gas used to the miner.	
	c)	Describes the types of values the EVM can interpret when writing Solidity code. The types of values the EVM can interpret are : **Booleans** Known in code as bool, the Booleans are true/false expressions that evaluate to true or false. **Signed and Unsigned Integers** Known in code as int and uint, these are numbers. They can be negative if they have a sign, or minus, indicating they are signed. Unsigned integers are thus positive numbers. **Addresses** The address type holds a 20-byte value, which is the size of an Ethereum address (40 hex characters, or 160 bits). Address types also have member types. **Members of Addresses** These two members allow you to query the balance of an account, or to transfer ether to an account. Be careful with transfer in smart contracts. It's better to use a pattern where the recipient is allowed to withdraw the money, than to have a contract initiating transfers. • balance • transfer **Address-Related Keywords**: Keywords come with the Solidity language. They are methods, so to speak, for using the language in predetermined ways. You can use these keywords in your code to accomplish common tasks needed in smart contracts. These include the following: • <address> .balance *(uint256):* Returns the balance of the address in wei • <address>.send (uint256 amount) returns (bool): Sends given amount of wei to address, and returns false on failure • this(current contract's type): Explicitly converts to the address • selfdestruct(address recipient): Destroys the current contract, sending its funds	

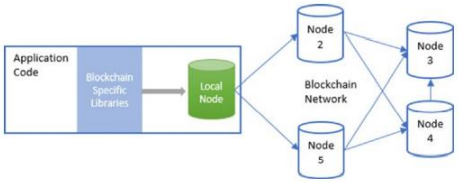
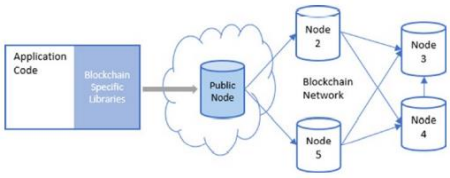
		to the given address.	
3	**Attempt *<u>any one</u>* of the following:**		**6**
	a)	Enlist and explain Hyperledger open source frameworks and tools. Hyperledger contains the following main open source frameworks and tools. – **Hyperledger frameworks:** • **Hyperledger Fabric (contributed by IBM):** This is a permission blockchain infrastructure with SDKs for Node.js, Java, and GoLang. Hyperledger Fabric is the heart of Hyperledger and supports chaincode in GoLang and JavaScript (utilizing Hyperledger Composer or natively). Blockchain is based on the endorser/orderer architecture •**Hyperledger Burrow**: This is an Ethereum VM built to specification. • **Hyperledger Indy**: Think independent. This is a tool and library for running independent identities on distributed ledgers. • **Hyperledger Iroha**: This is focused on mobile applications; the code is based on Hyperledger Fabric. • **Hyperledger Grid:** This is a solution for a supply chain on a distributed ledger. The framework encapsulates Hyperledger implementations of data types, models, and smart contracts as well as showcases practical ways to create a supply chain business solution. • **Hyperledger Sawtooth (contributed by Intel):** This framework includes dynamic consensus and enables hot swapping of consensus algorithms on a running network. This is a more traditional blockchain architecture. **Hyperledger tools:** • **Hyperledger Caliper:** This is a blockchain benchmark tool. • **Hyperledger Cello:** This is an on-demand blockchain module toolkit for creating, managing, and terminating blockchains. • **Hyperledger Composer**: This tool has collaboration features used with Hyperledger Fabric for building blockchains aimed at businesses for chaincode and blockchain applications. •**Hyperledger Explorer:** This is a module to view, invoke, deploy, and query blocks, transactions, and network data. • **Hyperledger URSA:** This is a shared cryptographic library; it includes shared projects such as the implementation of several different signature schemes • **Hyperledger Quilt/Interledger.js:** This is an Interledger Protocol (ILP), meaning an atomic swapping between ledgers. The payments protocol enables transferring an asset (value) across distributed and non distributed ledgers. There are two implementations: the Java one is called Quilt, and the JavaScript one is called Interledger.js.	
	b)	What are the components that make up Hyperledger Fabric network? A Hyperledger Fabric network consists of the following components: – **Assets:** Assets are key-value pairs that represent a value. A value can be anything such as a document, stock, or cryptocurrency token. Each asset holds a state and ownership.	

		Shared ledger: A shared ledger holds its own copy of the ledger with the state of the asset. This ledger is called the world state. The shared ledger also holds a copy of the blockchain, which stores the ownership of the asset by recording the transaction's history. **Smart contracts (chaincode):** Hyperledger Fabric calls smart contracts chaincodes that can be programmed in Go (GoLang) or JavaScript (Node.js). Chaincode can interact with the shared ledger, assets, and transactions and contains the business logic and can set an endorsement policy. **Membership services provider (MSP):** The MSP is the certificate authority that manages the digital certificate; it manages user IDs and authenticates all participants on the network. All members must be known identities in order to transact on Fabric. That's because the network is private and based on permissions. The MSP is used to authenticate and validate these members' identities and permissions. The MSP uses a certificate generation tool called cryptogen. **Peer nodes:** The Hyperledger Fabric network is built on peer nodes that are owned and contributed by members of the network. A node can be an organization or an individual. Nodes hold shared ledgers and can execute chaincode. Nodes can access ledger data; they can endorse transactions and interface with applications. Nodes can have permission to endorse peers or role for endorsers. Peer nodes receive ordered ledger state updates as part of the blocks they receive in order to maintain the ledger, or what Hyperledger calls world state. **Channel:** Channels can be created by a collection of peer nodes. A group of nodes can create a separate ledger of transactions. A channel is similar to the P2P channel you created when you formed your own blockchain. **Organizations**: Each peer node contributes resources, and together they form the collective network. The owning organization can assign peer nodes using a digital certificate through the MSP. Additionally, peer nodes from different organizations can join a channel. Organizations with separate peer nodes are able to share the same MSPs. Best practice is to have one MSP for each organization. **Ordering service:** This service packages transactions into blocks. Blocks can then be broadcast to peer nodes and clients on the shared P2P channel. The channel outputs the same messages with the same logical order to all peer nodes. A consistent logical order is called atomic delivery.	
	c)	Write a short note on "Tokens Are a Category of Smart Contract". Tokens are just one application of smart contract functionality on the EVM. Ethereum does make provisions for one common use-case of smart contracts, which is a subcurrency, a.k.a. token. In the hopes of making it easy to get up and running, the Ethereum developers have put an easy-to-use template inside the Mist wallet for quickly launching your own tokens. Presumably, other templates for common smart contracts will follow. But at present, the one we get out of the box is the ability to create a custom unit of value which can be passed around, alongside ether, within the EVM. **Tokens as Social Contracts** : Tokens are sometimes called coins, tokens themselves are smart contracts. But	

		tokens themselves (like all forms of money) can also be seen as social contracts, or agreements between groups of users. In plain English, the implicit agreement of a group using a token would be as follows: "We all agree this token is money in our community." It's also a tacit agreement not to counterfeit, undermining the system! The closest thing we have to a social contract in software form today is probably the end-user license agreements, or EULAs, that users sign when they create an account on services such as Facebook, Twitter, iTunes, or Gmail. This agreement usually includes language barring activities such as spamming other users, which would degrade the user experience. **Tokens Are a Great First App** When making a token, consider that it is only as valuable as the community using it believes it will be. Thus, it is far easier to launch a token into an existing community that already trades using some kind of money or scrip. However, making sub currencies is not the only use of a cryptoasset. The concept of an asset is highly generalized. Assets, in the form of financial contracts or smart contracts, can be used to represent shares of equity, or lottery tickets, or just scrip within a local economy. In Ethereum, tokens exist within, and rely upon, the public blockchain: you can create a subcurrency of ether, but ether will always remain the priviliged token with which miners and gas costs are paid. If you want a purely independent blockchain network, you can create your own private blockchain and be completely disconnected from the main Ethereum chain.	
4	**Attempt *<u>any one</u>* of the following:**		**6**
	a)	Write a short note on "DAG & Nonce" under mining ether. In effect, each node is playing a guessing game with itself, trying to guess a nonce that will validate the current block; if it guesses the right nonce, it wins the block reward. If not, it continues guessing until it gets word that another node on the network has found a winner. Then, it discards the block it was mining downloads the new block, and begins mining a new block on top of that one. But the node gets both parameters of the guessing game, as well as a new pair of dice (so to speak) with each potential block as it rolls in. The rules of the guessing game are designed this way to prevent clever individual nodes from outsmarting the system in the pursuit of more mining rewards. Therefore, you can think of the DAG file as a way of standardizing the solution time of the proof-of-work algorithm. It levels the playing field for miners, but more important, helps cluster block times around the 15-second mark by ensuring that—even with massive computing power—you can't guess the correct nonce a whole lot faster than your competitors. All the data a node needs to participate in the guessing came is drawn from the blockchain itself. In cryptography, an encryption seed can be used to help generate a pseudorandom number, thus increasing the randomness of whatever encrypted output the Ethash algorithm produces. In Ethereum and Bitcoin, each node gets the seed from looking at the hash of the last known winning block. In this way,	

		the node must be mining on the correct, canonical chain in order to play the game correctly. Performing proof of work on an erroneous block (say, an uncle) cannot yield a winning block. This is helpful if you're trying to reduce unfair advantage in a proof-of-work scheme, which could be used by a large pool of miners to highjack the network onto a version of the truth in which everyone's ether is transferred to the hijacker's accounts. Here is the process by which a node sets itself up to perform the PoW guessing game: 1. From an encryption seed derived from the block header, the mining node creates a 16 MB pseudorandom cache. 2. In turn, the cache is used to generate a larger 1 GB dataset that should be consistent from node to node; this is the DAG. This dataset grows over time, in a linear fashion, and is stored by all full nodes. 3. Guessing the nonce requires the machine to grab random slices of the DAG dataset and hash them together. This works similarly to using a salt with the hash function. In cryptography, a random data chunk you toss into a one-way hash function is called a salt. Salts are like nonces: they make things more random, and thus more secure.	
	b)	What are the seven steps only after which a block canonized as valid and true? In order to escape uncle-hood and become the heaviest block, a true block (sometimes called a nephew) needs to pass muster with a long series of steps used in the processing of each block. An important component of this process is the block validator algorithm. This algorithm seeks to validate the hash that comes with the block, located in the block's header. This aspect of block processing makes a good on-ramp to the anatomy of a block as a data object. Before a completed block can undergo processing and acceptance by the rest of the network, and before nodes can begin mining on top of a new block, each and every node must independently download and validate the block before beginning to mine in top of it. Here are all the steps the block validator algorithm takes, in order: 1. Check if the previous block referenced exists and is valid. 2. Check that the timestamp of the block is greater than that of the referenced previous block and less than 15 minutes into the future. 3. Check that the block number, difficulty, transaction root, uncle root and gas limit (various low-level Ethereum-specific concepts) are valid. 4. Check that the nonce on the block is valid, showing the evidence of proof of work. 5. Apply all transactions in this now-validated block to the EVM state. If any errors are thrown, or if total gas exceeds the GASLIMIT, return an error and roll back the state change. 6. Add the block reward to the final state change.	

		7. Check that the Merkle tree root final state is equal to the final state root in the block header. Only after these seven steps is a block canonized as valid and true!	
	c)	What is cryptoeconomics? Enlist the domains of cryptoeconomics. Why cryptoeconomics is useful? To secure the information they send across networks, today's computers can encrypt information with far greater strength than the Enigma machine circa 1945. Cryptographic messaging can be loosely defined as communication in an untrustworthy environment, or under any circumstances where your information is prone to exploitation or destruction. War is one example, but so are industrial espionage, religious persecution, or even natural disasters. The field of economics typically studies interactions between people, sometimes in hostile contexts such as war. The emerging field of cryptoeconomics is the study of economic activity conducted across network protocols in an adversarial environment. The **domains** of cryptoeconomics include the following: • Online trust • Online reputation • Cryptographically secure communication • Decentralized applications • Currency or assets as a web service (so to speak) • Peer-to-peer financial contracts (smart contracts) • Network database consensus protocols • Antispam and anti-Sybil attack algorithms **Cryptoeconomics is Useful :** Applied cryptoeconomics is about engineering a layer of defense between public networks and attackers of all sizes. It combines game theoretical system design, encryption, and cryptographic hashing to protect a commonly used, commonly operated resource —in this case, a global transaction state machine. Because public chains are public, they need to be resilient against attackers with large amounts of computing power. Hence, networks with more nodes, and more geographically distributed nodes, owned by discrete unlinked owners, are considered more secure. Mining pools contribute to centralization, which is why any pool with larger than 25 percent hashpower is approaching the threshold of network threat. Should two such pools emerge, they might quickly get control of a network. By using the custom, ASIC-resistant Ethash algorithm and designing the network to quickly increase in difficulty, the protocol designers ensured there would be little incentive for miners to professionalize and consolidate.	

5	Attempt ***any one*** of the following:		6
	a)	Explain with diagram Decentralized Application Architecture. The decentralized applications are meant to directly interact with the blockchain nodes without the need for any centralized components coming into picture. However, in practical scenarios, with legacy systems integrations and limited functionality and scaling of the current blockchain networks, sometimes we must make choices between full decentralization and scalability while designing our DApps. **Public Nodes vs. Self-Hosted Nodes** Blockchains are decentralized networks of nodes. All nodes have the same copy of data and they agree on the state of data always. When we develop applications for blockchains, we can make our application talk to any of the nodes of the target network. There can be mainly two set-ups for this: • **Application and node both run locally**: The application and the node both run on the local machine. This means we will need our application users to run a local blockchain node and point the application to connect with it. This model would be a purely decentralized model of running an application. An example of this model is the Ethereum-based Mist browser, which uses a local geth node  Figure: DApp connets to local node • **Public node:** The application talks to a public node hosted by a third party. This way our users don't have to host a local node. There are several advantages and disadvantages of this approach. While the users don't have to pay for power and storage for running a local node, they need to trust a third party to broadcast their transactions to the blockchain. The Ethereum browser plugin metamask uses this model and connects with public hosted Ethereum nodes.  Figure: DApp connets to public node	
	b)	Explain how to set up a Private Ethereum Network? To set up a private Ethereum network, we will need one of the many Ethereum clients available. In simple terms, an Ethereum client is an application that implements the Ethereum blockchain protocol. There are many Ethereum clients available on the Internet today; one of the popular ones is go-ethereum,	

also known as geth. We will be using geth for our private network set-up.

Install go-ethereum (geth)

The first step is to install geth on our local machine. To install geth, we will get the geth executable installer from the official source. Download the installer package for your platform and install geth on your local machine. You can also choose to install geth on a remote (cloudhosted) server/virtual machine if you do not want to install it on your local machine. Once geth is successfully installed on your local machine, you can check the installation by running the following command in your terminal/command prompt.

```
geth version
```

Create geth Data Directory

By default, geth will have its working directory but we will create a custom one so that we can track it easily. Simply create a directory and keep the path to this directory handy.

```
mkdir mygeth
```

Create a geth Account

The first thing we need is an Ethereum account that can hold Ether. We will need this account to create our smart contracts and transactions later in the DApp development. Enter and confirm the passphrase and then your geth account will be created. Make sure to remember the passphrase you entered; it will be needed to unlock the account later to sign transactions.

Create genesis.json Configuration File

After installing geth and creating a new account, the next step is to define the genesis configuration for our private network. As blockchains have a genesis block that acts as the starting point of the blockchain, and all transactions and blocks are validated against the genesis block. For our private network, we will have a custom genesis block and hence a custom genesis configuration. This configuration defines some key values for the blockchain like difficulty level, gas limit for blocks, etc.

Run the First Node of the Private Network

To run the first node of the private blockchain, let's first copy the JSON from the previous step and save it as a file named genesis.json. For simplicity, we are saving this file in the same directory that we are using as the data directory for geth

Run the Second Node of the Network

There is no network with just one node; it should at least have two nodes. So, let's run another geth instance on the same machine, which will interact with the node we just started, and both these nodes together will form our Ethereum private network. To run another node, first of all we need another directory that can be set as the data directory of the second node.

	c)	How an EVM backend talks to a JS frontend ? The gap between the Ethereum network and what might be called the HTTP network, otherwise known as the Web, can indeed be traversed. Let's say a customer enters a lunch order on a dapp-powered web site from a conventional web browser. In order to successfully pass data about her order (how many milkshakes?) between her browser and the EVM, the dapp's front end must "send" the data to the EVM in a certain format. In computing, data-interchange formats work much like the international postal service. Although different servers around the world may be running different operating systems, written in different languages, by totally different minds, they must at some point exchange data with a server that is not like them. To get the "translation" correct, programmers engineer their programs to send information to other programs in a certain notations. Usually, the notation describes a format for an entire object (a set of attributes and values). For example, a human data object might include height, weight, eye color, foot size, and so on.. **JSON-RPC**: In today's web applications, JavaScript code can pass information across the Web by using a common object notation called JavaScript Object Notation (JSON). JSON objects can contain numbers, strings, and ordered sequences of values for certain attributes. There are two important data objects in Web3.js, which are roughly equivalent to JSON in the way they are passed between the front and back ends of an Ethereum-powered application. They are called JSON-RPC objects and they come with the Web3.js library. These two objects are used in the following ways: • web3.eth is used specifically for blockchain interactions. • web3.shh is used specifically for Whisper interactions. Whisper is a private messaging protocol that is itself a part of the larger Ethereum protocol. Thus, JSON-RPC objects works as passing back and forth constantly between the front end (on the HTTP Web) and the back end (the Ethereum Web). Web 3 is a general term for the decentralized web, just as Web 2 was defined by webhosted applications and services. Web 1 refers to the original World Wide Web, which hosted static pages. Web 3 is very much a vision that centers on the Ethereum protocol in particular. It is generally considered to have three components: Peer-to-peer identity and messaging system Shared state (a blockchain) Decentralized file storage	