Blockchain Technologies 1

Assignment - 1

Aibek

SE-2419

Contents

Module 1: Introduction to Blockchain Technology

1. Distributed vs. Centralized Ledgers

Explain how a distributed ledger differs from a centralized ledger in terms of trust, confidentiality, fault-tolerance, and attack surface. Provide at least 3 real-world examples for each.

Answer:

The essential discrepancy between distributed ledger (DL), commonly represented by blockchain technology, and centralized ledger (CL), usually a traditional database, is their architecture and the mechanisms they use for data integrity and coordination. A centralized system has clients that are connected to a single central server governed by an administrator, while a distributed ledger network is based on the peer-to-peer model where control as well as data distribution occur across many nodes.

Trust

Centralized Ledger:

- **Trust Model**: Centralized systems rely on a trust-based model. Trust is implicitly or explicitly placed in a single central authority, administrator, or intermediary (like a bank) who manages the entire system and controls the data
- **Technical Reasoning**: The integrity and authenticity of the ledger are maintained exclusively by this single entity, meaning there is no technical guarantee against malicious actions by the controller

Distributed Ledger:

- **Trust Model**: Distributed ledgers operate on a trustless model (or distributed trust). Trust is established and maintained through cryptographic security and a consensus mechanism rather than relying on a single third party,,. Participants collectively agree on the state of the network.
- **Technical Reasoning:** Consensus protocols (like Proof-of-Work or Byzantine Fault Tolerance variants) ensure that new records are added only if participants collectively agree to do so. Transparency, where all participants possess the same verifiable information, reinforces this trust among participants
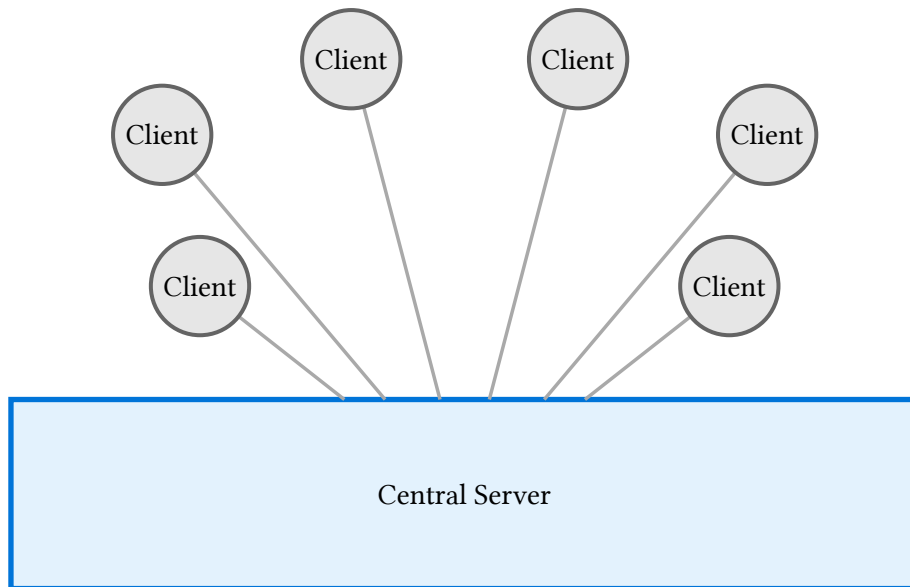
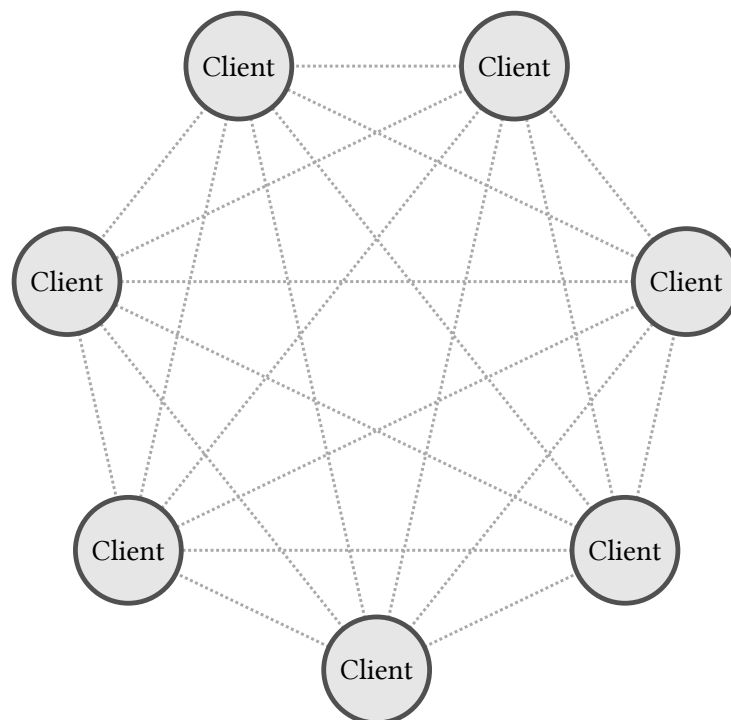Figure 1: Centralized Ledger (CL): Single Point of Control

Figure 2: Decentralized Ledger (DL): No Single Point of Control

Confidentiality

Centralized Ledger (CL):

- **Data Access:** The central authority is responsible for maintaining confidentiality and the access control policies established by it are the means through which this is done. The data is so to speak owned by the controlling entity, and access is limited to a few select individuals.
- **Technical Reasoning:** The central server's data security utilizes traditional methods such as authentication, access control, and physical security measures.

Distributed Ledger (DL):

- **Data Access:** Confidentiality is a big issue and very different for each type of DL:

- ▸ Public (Permissionless) DLs has the main concern of transparency, thus all transactions are recorded in a shared, publicly available digital ledger. The user identities are usually anonymous but the transaction data is generally public.

  - ▸ Private/Consortium (Permissioned) DLs have the access limited only to the known participants, thus providing both confidentiality and transparency only within that group.

- **Technical Reasoning:** Cryptography is the main source of security for confidentiality and uses methods like encryption to the communication links (in transit) or the data when it is stored (at rest)

Fault-tolerance

Centralized Ledger (CL):

- **Resilience:** Centralized systems have very low fault tolerance as it is based on the one and only central server. This is exactly what a single point of failure means.
- **Technical Reasoning:** When the only central node ceases to function, the whole database becomes user-unreachable. The system is stuck because there is no distributed copy or coordinated failover, which is not the case since the core design is not supporting it.

Distributed Ledger (DL):

- **Resilience:** High fault-tolerance is the very property of distributed ledgers. The reason is that the data being distributed over many nodes, and the system is still available and on-going if one or more nodes go down.
- **Technical Reasoning:** Replication is the way to bring about fault tolerance. Typically, distributed systems that require consensus operate based on strict thresholds to guarantee safety, such as $N \geq 3F+1$ for tolerating Byzantine faults (where N is total nodes and F is faulty nodes). The system achieves liveness as long as a majority or a sufficient quorum of non-faulty nodes remains up and running and continues to process requests

Attack Surface

Centralized Ledger (CL):

- **Vulnerability:** The entire attack surface is concentrated in one single point-the central authority.
- **Technical Reasoning:** In total, an invader aspiring to disrupt, corrupt, or freeze the system will only have to take over the central node (the database, API endpoints, or the governing administrator)together with the system. Once the point of failure is breached, the hacker has full control of the data and system operations,.

Distributed Ledger (DL):

- **Vulnerability:** The network's overall security is based on the fact that all participating nodes are the points of attack. Through cryptographic hashing, the data is secured so that a chain is formed which is almost impossible to alter and which has very high resistance to penetration through tampering.

- **Technical Reasoning:** An adversary who wants to execute a devastating attack must first break the economic barriers and seize the control of a considerable part of the network which usually means a supermajority or more than 50% of the computational power (in Proof of Work networks like Bitcoin) or over one-third of the total validator stake/nodes (in BFT/PoS networks). Changing a record is practically impossible because its hash is linked to consecutive

blocks, hence altering one record requires recalculating the hashes of all subsequent blocks, making data tampering-proof.

Feature	Centralized Ledger	Distributed Ledger
Trust	Relies on a single authority	Trust is distributed among nodes
Confidentiality	High (Owner controls access)	Varies (Public vs. Permissioned)
Fault-tolerance	Low (Single point of failure)	High (Redundancy)
Attack Surface	Central server is the target	Consensus mechanism/Nodes

Table 1: CL vs DL (feautres)

Ledger Type	Trust	Confidentiality	Fault-Tolerance
Centralized	Traditional Banking Systems (like SWIFT, single bank ledgers)	Corporate ERP Systems (data managed by the company)	Traditional Web2 Applications (for example, a service running on one cloud server/database)
Distributed	Bitcoin (Public, permissionless network based on Proof-of-Work)	Ethereum (Public DApp platform using smart contracts and consensus)	Consortium Blockchains (like supply chain solutions involving several organizations using permissioned networks)

Table 2: Real-World Examples

2. Definition of Immutability

Provide a rigorous technical definition of immutability. Explain how hash functions contribute to this and describe one scenario where immutability fails.

Answer:

Immutability in a blockchain refers to an unalterable and non-removable transaction after it has been confirmed and included in a block that belongs to the canonical chain. The ledger is only for adding new transactions: newly processed blocks can only be added one after another, and the entire history is kept permanently. Mistakes are not rectified by replacing them in the past, but rather with the addition of new reversing transactions.
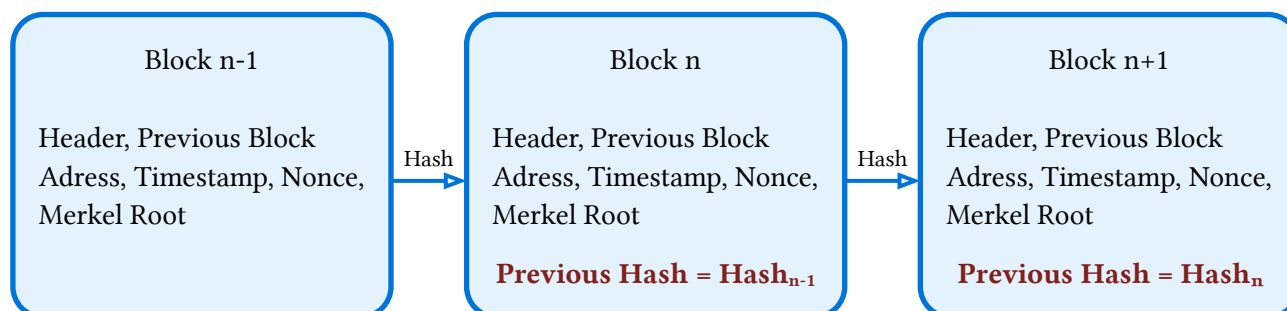
How Hash Chaining Ensures Immutability The link between the blocks is established with the cryptographic hash of the preceding block:

An alteration of even 1 bit in a past block dramatically changes its hash (avalanche effect).

This leads to the "previous hash" reference in the next block being broken.

To cover up the tampering, a hacker needs to perform the same intense computing work that involves re-mining the compromised block plus all the consecutive blocks.

In a Proof-of-Work system, the massive power cost needed makes this practically impossible.

Block n-1		Block n		Block n+1
Header, Previous Block Adress, Timestamp, Nonce, Merkel Root	Hash →	Header, Previous Block Adress, Timestamp, Nonce, Merkel Root **Previous Hash = Hash$_{n-1}$**	Hash →	Header, Previous Block Adress, Timestamp, Nonce, Merkel Root **Previous Hash = Hash$_n$**

Blockchain Hash Chaining – Enforcing Immutability

Figure 3: Each block stores the hash of the previous block (in red). Modifying any block changes its hash, breaking the chain.

Immutability isn't absolute—it's economic finality. Suppose an adversary gains control of the hash power of the network that is more than 50%.

They first make the transaction public and then create a hidden chain that does not include that transaction and is longer than the public one. When the hidden chain is revealed, the participants in the network apply the longest-chain rule and accept it. The time of the original transaction is then erased from the canonical history → successful double-spend.

In this way, the immutability of the blockchain is guaranteed by cryptography but the costs that have to be borne for acquiring majority hash power finally determine the security.

3. Transparency vs. Privacy
Evaluate blockchain transparency vs. privacy.
- Compare Bitcoin vs. Ethereum
- Explain mixers, stealth addresses, and ZK proofs.

Answer:

Blockchain Transparency vs. Privacy
Blockchain technology in its very nature supports transparency. Transparency is considered the primary attribute since the global digital ledger used for recording transactions is accessible and verifiable by all. This common and checkable data creates trust between the parties and ensures that there is always evidence because all the information is capable of being traced. Thus, transparency together with immutability creates an environment that does not need the involvement of third-party intermediaries which results in lower costs and faster processing.

The conflict between transparency and privacy is evident; on one hand, the decentralization of the network renders intermediaries unnecessary, on the other hand, the requirement for digital cash to be decentralized implies that both accountability (the prevention of double-spends) and anonymity (the grant of privacy) have to be dealt with. Regardless, the very nature of blockchain guarantees that each transaction is public and can be verified. In a public blockchain scenario,

which is characterized by wide geographical distribution and lack of trust, there might be some bad actors that try to listen in; therefore, in these situations, encryption is the standard method used to ensure confidentiality.

Comparison of Bitcoin and Ethereum

When Bitcoin and Ethereum are fundamentally different in their approach, in terms of both purpose and consensus mechanism, one finds the real point of comparison coming with transaction throughput, speed, and other performance metrics.

	Bitcoin	Ethereum
Purpose	A credible alternative to traditional fiat currencies (medium of exchange, potential store of value)	A platform to run programmatic contracts and applications via Ether
Consensus	Proof-of-Work (PoW)	Proof-of-Stake (PoS) (as per forecast, reflecting the transition)
Transaction Model	Unspent Transaction Output (UTXO). Your balance is the sum of all distinct, unspent outputs, which must be spent entirely in a single transaction	Account-based model (Externally Owned Accounts and Contract Accounts),. An account holds a single balance that increases or decreases
Block Time	10 minutes on average	12 seconds on average
TPS	3-7 transactions per second	10-30 transactions per second
Supply	Finite supply, capped at 21 million BTC	No fixed maximum supply; issuance to validators but some ETH is burned, sometimes making supply net-deflationary

Table 3: Bitcoin vs Ethereum

Mixers

Services or protocols that break the on-chain link between source and destination of funds.

Mixers take the coins from a lot of users and mix them up so that it is hard to identify which output is from which input. For Bitcoin, this is usually done through collaborative transactions (like CoinJoin), whereas for Ethereum, mixers mostly rely on smart contracts that take deposits and later let people withdraw to a new address using a secret. On the one hand, mixers enhance privacy; on the other hand, they can create compliance and legal issues if used to hide illegal money.

Stealth address

Stealth addresses generate unique, one-time payment addresses through cryptographic key agreements. This ensures the recipient's privacy by making it impossible to trace the connection between their public identity and on-chain transactions.

Stealth addresses hide the connection between a receiver's public identity and the real address that receives funds. Thus, they protect the privacy of the recipient rather than that of the sender or the amount. The usual scenario is that the receiver discloses some long term public key data, the sender and the receiver produce a one-time payment address using a Diffie–Hellman like key agreement and a derivation scheme, which makes it appear to the blockchain as if the recipient is using a new random address that is not easily associated with them. This provides a level of privacy comparable to using a new address for each payment but with on-chain derivation and optional scanning keys, so only the recipient can recognize and spend from these stealth outputs.

ZK proofs and privacy

Zero-knowledge (ZK) proofs let a party prove that "this transaction or statement is valid under the rules" without revealing underlying data such as which exact notes are spent, which address is used, or what attribute values are. On Ethereum, ZK is used in several ways:

- Mixers like Tornado Cash: prove "I own one of the deposits in this Merkle tree" without revealing which leaf.

- ZK-rollups and private payment/DEX systems: prove correctness of batched transactions while hiding individual details.

- ZK identity: prove properties such as age, membership, or ownership of a credential without disclosing the identifier or full record, enabling Sybil-resistant yet privacy-preserving identity.

Overall, Bitcoin's base layer leans heavily toward transparent auditability with bolt-on privacy tools, while Ethereum's programmable environment exposes more activity but also makes sophisticated privacy primitives like mixers, stealth address schemes, and ZK systems first-class smart contract applications.

4. DApp Architecture

Define DApp architecture in detail.
Describe how the following components interact:
- Smart contract layer
- Off-chain backend
- Frontend
- Wallets (EIP-1193 providers)
- Nodes (RPC, full, light)

Answer:

A Decentralized Application (DApp) is a software program that runs on a blockchain or a peer-to-peer (P2P) network of computers and thus does not depend on centralized servers. DApps rely on smart contracts to carry out their core consumptions; in this way, all operations are decentralized, transparent, and secured by cryptography. One of the key differences between DApps and Web2

applications is the way they handle data and business logic: the former spreads them over various nodes while the latter concentrates them on one or more servers and thus gets rid of bugs, allows for constant improvements, etc., at the cost of centralization.

The DApp architecture is based on three main pillars: the frontend (user interface), the smart contract layer (business logic), and the blockchain (data storage).

Component Interactions in Transaction Flow

The transactions in a DApp come along with a structured flow starting from the user input to the on-chain execution, and this is the point where the seamless interactions among the components happen.

Frontend

The frontend user interface (UI) is the and is usually made using web technologies like HTML, CSS, and JavaScript frameworks (e.g. React). It handles the user actions like transfer and swap that happen with the help of button clicks and creates the transaction payload that calls the specific functions of smart contracts. In this part, there are no changes of states and it acts like a bridge that connects users to the blockchain with the help of libraries such as ethers.js or web3.js.

Wallets (EIP-1193 Providers)

Wallets are usually browser extensions like MetaMask and represent Externally Owned Accounts (EOAs) that are controlled by private keys and follow EIP-1193 standards for provider interfaces. When the frontend makes a request, the wallet asks for user approval, then signs the transaction with cryptographic methods, and finally estimates and pays the gas fee. The wallet then sends back the signed transaction to the frontend so it can be sent out.

Smart Contract Layer

Smart contracts (for example, on Ethereum) are an inherent part of the blockchain that contain immutable bytecode executing the business logic in the Ethereum Virtual Machine (EVM). When a signed transaction is available, the EVM acts on the called function in accordance with the rules laid down beforehand, modifies the global state (e.g. balances) and generates events for logging. The process is predictable and consensus is maintained among nodes.

Nodes (RPC, Full, Light)

Nodes are the major part of the P2P blockchain network maintainers; they store and validate the ledger.

Remote Procedure Call (RPC):

Acts as the API gateway, e.g., JSON-RPC via Infura or Alchemy, facilitating transactions and balance or event querying by the frontends and wallets.-

Full nodes:

Keep the complete chain history, run all transactions via EVM, block validation, and consensus enforcement.-

Light nodes:

Have less resource requirements, keep only block headers and apply Simplified Payment Verification (SPV) to connections with full nodes for querying without entire storage.

Step	Components involved	Action
1. Initiation	Frontend → Wallet	UI captures input; wallet signs tx fiveable+1
2. Broadcast	Wallet → RPC Node	Signed tx sent via RPC geeksforgeeks
3. Validation/ Execution	Full/Light Nodes → Smart Contracts	EVM runs logic; state updates fiveable
4. Confirmation	Nodes → Frontend/Off-chain	Polling/receipts update UI; events indexed geeksforgeeks

Table 4: Transaction Flow Summary

This architecture ensures trustless, verifiable operations ideal for applications like DeFi and NFTs.
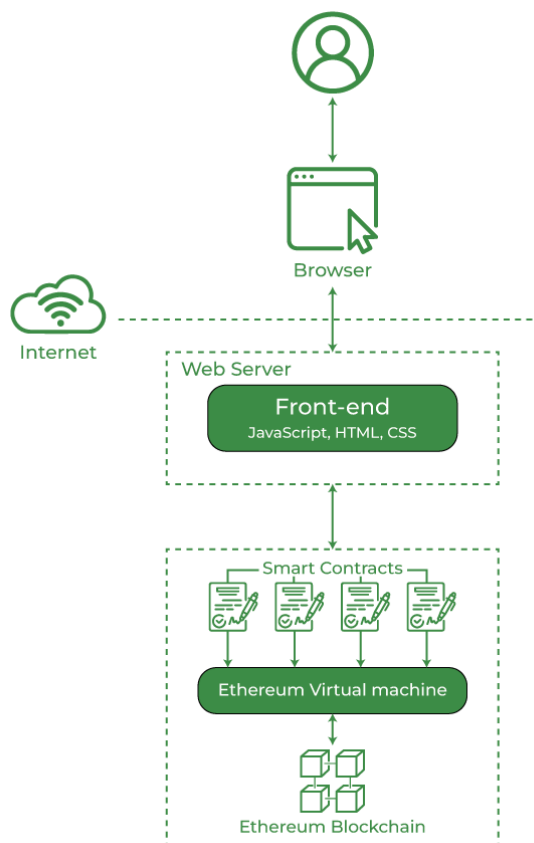

Figure 4: DApp Architecture diagram

The architecture of the Decentralized Application (DApp) is presented in an uncomplicated diagram (Figure 4) consisting of three layers: first, a user actions take through a frontend based on the browser, and then the frontend connects to the smart contracts running in the Ethereum Virtual Machine (EVM) on the Ethereum blockchain.

Moving from the top to the bottom:

- First and foremost, the user through a browser gets the frontend (HTML, CSS, JavaScript) from a web server over the Internet.

- Then the frontend sends requests (such as sending transactions or calling functions) to the smart contracts that have already been deployed.

- Finally, the execution of the smart contracts takes place in the Ethereum Virtual Machine, which consequently updates and reads data from the Ethereum blockchain thus achieving a decentralized storage of contract state and transaction history.

Module 2: Cryptography Fundamentals

1. SHA-256 Computations

Compute SHA-256 hashes using at least two tools

Answer:

1. Node.js Code:

```javascript
const crypto = require('crypto');

function createSHA256Hash(inputString) {
  const hash = crypto.createHash('sha256');
  hash.update(inputString);
  return hash.digest('hex');
}

const myString = 'SE-2419';
const hash = createSHA256Hash(myString);
console.log(hash);
```

1. Node.js Output:

```shell
37e9bcf9787d084d18b69f2094995c80617ce56116897fe903abc120f6dc83c8
```

2. Online hashing tool

Figure 5: Screenshot of online hashing website

13

3. Linux Terminal:

```
aibek@LAPTOP-CKRHQPN1:~$ printf "SE-2419" | sha256sum
37e9bcf9787d084d18b69f2094995c80617ce56116897fe903abc120f6dc83c8   -
```

```shell
1  prinitf "SE-2419" | sha256sum
2  37e9bcf9787d084d18b69f2094995c80617ce56116897fe903abc120f6dc83c8  -
```

Listing 2: Code

2. Comparison

Write a comparison of the outputs, and explain why all must be identical despite different tools.

Answer:

The output of the computed SHA-256 hash of the string "SE-2419" was the same on all three platforms: Node.js, the online hashing tool, and the Linux `sha256sum` utility.

The uniformity of the output is one of the major characteristics of cryptographic hash functions and is determined by the idea of determinism.

Deterministic Algorithm:
SHA-256 is a universally accepted deterministic mathematical algorithm that operates through the application of a standard. Consequently, the hash function for any particular input sequence of bits will always output a fixed-size 256-bit output. It does not matter which tool, programming language, or operating system is employed for the calculation as long as the strict following of the Secure Hash Standard (NIST FIPS 180-4) is ensured.

The Role of Byte Stream: n If the outputs were not the same, the most likely cause of the difference would be a variation in the input byte stream. For instance, one tool might have included a hidden character (like a null terminator or a space) that the other tools did not. But since the input byte stream for "SE-2419" was the same for all methods, the hash output was the same: 37e9bcf9787d084d18b69f2094995c80617ce56116897fe903abc120f6dc83c8

3. Collison resistance

- Attempt to modify one bit of input
- Show how drastically the hash changes
- Explain why finding two identical SHA-256 hashes is computationally infeasible
- Estimate the probability of collision using the birthday paradox formula

Answer:

In a bid to showcase the collision resistance property of the SHA-256 algorithm, we will take the original input "SE-2419" along with its hash and then carry out a minimal alteration on the input by appending a newline character, "n", which is a minor change of only one byte in the sequence adding a single byte of value 0x0A. This can be seen as a "one-bit" or minimal perturbation, although the adding of "n" alters 8 bits, however, it shows the avalanche character, which means that even tiny input changes produce outputs that are completely different. We will compute the

hashes again with the same tools but with changes as per the specifications: in Node.js, change the input to "SE-2419\n"; in Linux, use `echo` instead of `printf`, which adds a trailing newline by default.

Original Input and Hash (for Reference)

Input: "SE-2419" (no trailing newline)

Hash: 37aeb679f9780b4d1b6f5f2949905c80617ce56116897fe98a3bc12b6fdc3bc8

Modified Input: "SE-2419\n" (with trailing newline)

Now, recompute with each tool:

Node.js Code:

Changed the input to "SE-2419\n" in Code Block 1-9 to

```javascript
const myString = 'SE-2419\n';
```

Node.js Output:

```shell
f03d8f4d66a3fbcb9ee6fd3a580dca0624cbd8f6ec868e3890de4faeef71518f
```

Linux Terminal (using echo "SE-2419" | sha256sum):

```
aibek@LAPTOP-CKRHQPN1:~$ echo "SE-2419" | sha256sum
f03d8f4d66a3fbcb9ee6fd3a580dca0624cbd8f6ec868e3890de4faeef71518f  -
```

```shell
echo "SE-2419\n" | sha256sum
f03d8f4d66a3fbcb9ee6fd3a580dca0624cbd8f6ec868e3890de4faeef71518f  -
```

How Drastically the Hash Changes

The hash that was not altered begins with "37aeb679…" and finishes with "…6fdc3bc8". On the other hand, the modified hash (which is the result of adding just "\n") will not at all resemble the original hash, for example, it will start with "f03d8f4d…" and there will be no similarities recognized among the 64 hexadecimal characters. The reason for this is the avalanche effect of SHA-256, which means that output bits of every input bit pass through many rounds of mixing (pseudorandomness functions such as bitwise operations, additions, and rotations) and they all get entangled. The output is different by almost 50% in the case of a single bit flip (or 8 bits added in the case of "\n").

Why Two Identical SHA-256 Hashes Cannot Be Found Computationally

A collision happens when two distinct inputs result in the same hash output. The collision resistance of SHA-256 is due to the fact that it has a 256-bit output space, which means there are 2^{256} possible hashes (around 1.1579×10^{77} unique values, which is a number larger than the atoms in the visible universe). In order to find a collision intentionally (preimage or second preimage attack), the hacker would have to brute-force search a vast number of inputs, which would require computational power that is way beyond what is available with current technology. Even if quantum computers were used (through Grover's algorithm), $O(2^{128})$ operations would still be required for a preimage attack, which is not practical with the hardware that is expected to be available in the near future. SHA-256 does not have any known practical collisions (in contrast to weaker hashes like MD5), and its design includes resistance to differential cryptanalysis and other attacks.

Brithday Paradox Formula

The birthday paradox is a classic example that demonstrates the extent to which our intuition can fail when it comes to the amount of 'birthday' cases or collisions, as they are called, arising in small groups of people or in this case samples of random numbers.

For the case of SHA-256, assume the hash space as the "birthdays" where there are d = 2^{256} possible values. The formula for the probability p of having at least one collision in n random hashes is: $p \approx 1 - e^{-\frac{n^2}{2d}}$

When p = 0.5 that is, for the case of a 50% chance of collision, we need to find n now: $n \approx \sqrt{2d \times \ln(2)} \approx 1.177 \times \sqrt{d} = 1.177 \times 2^{128} \approx 4 \times 10^{38}$

Thus you would need to produce approximately 2^128 (340 undecillion) hashes to get a 50% chance of a random collision which requires a huge energy input.

Module 3: Developer Environment Setup

Activity Requirements

Set up a blockchain environment (Node.js, npm, VS Code). Initialize a project and install `web3`, `ethers`, `crypto-js`.

Answer:

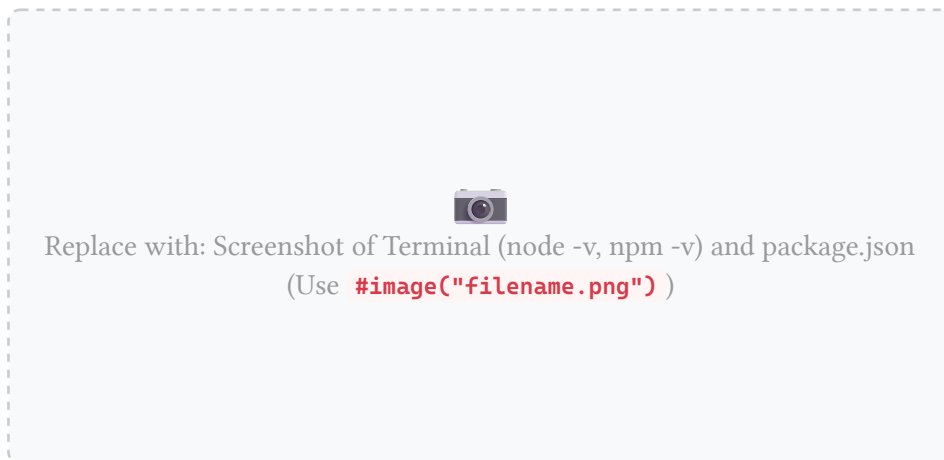**Package Installation & Version Check:**

Replace with: Screenshot of Terminal (node -v, npm -v) and package.json
(Use `#image("filename.png")`)

Figure: Screenshot of Terminal (node -v, npm -v) and package.json

Package Explanation:

- **web3:** ...
- **ethers:** ...
- **crypto-js:** ...

Module 4: Models & Architecture

1. Bitcoin's UTXO Model

Draw a diagram of UTXO flow. Explain script validation, parallelism, and stateless validation.
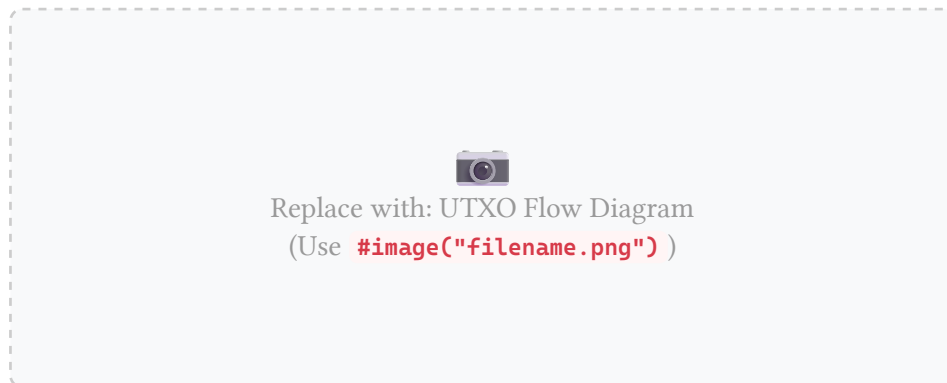
Answer:

Replace with: UTXO Flow Diagram
(Use #image("filename.png"))

Figure: UTXO Flow Diagram

Script Validation Steps:
1. ...
2. ...

2. Ethereum's Account Model

Explain EOA vs. Contract Accounts. Describe nonce, balance, storage, codeHash. Provide a JSON example.

Answer:

Account State JSON Example:

3. Security Implications

Analyze Replay vulnerabilities, Transaction malleability, Double-spend handling, Smart contract attack surface, and State bloat.

Answer:

4. EVM Architecture

Explain EVM bytecode execution, Stack-based model, Gas metering, and Error handling.

Answer:

5. Smart Contracts

Explain gas cost model and storage. Why are writes expensive? Provide code optimization example.

Answer:

Inefficient Code:

   Optimized Code:

   Explanation: ...

Module 5: Wallets & Transactions

Step 1: Wallet Installation

Install MetaMask/Trust/Rabby. Submit screenshot of interface and public address.
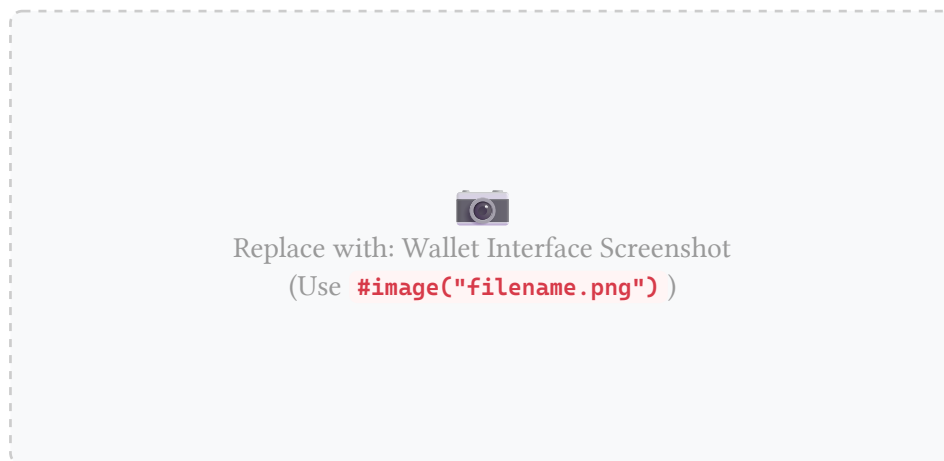
Answer:

Replace with: Wallet Interface Screenshot
(Use `#image("filename.png")`)

Figure: Wallet Interface Screenshot

Public Address: `0x ...`

Step 2: Transaction Inspection

Decode a raw Bitcoin or Ethereum transaction.

Answer:

Selected Blockchain: Ethereum/Bitcoin

Transaction Hash: `0x ...`

Decoded Fields:

Field	Value / Explanation
Nonce	...
Gas Price	...
Gas Limit	...
Value	...
Data/Input	...

To	...

Logic Explanation: