**Chapter 1: Introduction to Cryptography & Cryptocurrencies**

**1. Hash Function**

A **hash function** is a mathematical tool that does three things:

* **Takes any input**: No matter how small or large the input is, it can be hashed.
* **Produces a fixed-size output**: No matter how big or small the input, the result (output) is always the same size, like a 256-bit number (which is a long string of 256 ones and zeros).
* **Easy to compute**: Given an input, you can quickly calculate its hash output.

Think of a hash function as a machine that turns any size of text into a unique code of a fixed size. For example, you could input a short word like "cat" or an entire book, and the machine would spit out a code that's always 256 bits long.

**1.1. Cryptographic Hash Function**

A **cryptographic hash function** is a special kind of hash function used for security. For it to be secure, it must have three extra properties:

**1.1.1. Collision Resistance**

* **What it means:** It's really hard (almost impossible) to find two different inputs that produce the same hash output.
* **Example:** If you hash two different files (say, "file A" and "file B"), it should be nearly impossible for them to result in the same hash code.

However, there's a problem called the **Birthday Paradox**, which says that if you try enough random inputs, you'll eventually find a collision. For a 256-bit hash, even though there are a huge number of possible outputs, the chances of finding a match increase significantly if you try enough inputs. This is because of how probabilities work.

**1.1.2. Hiding**

* **What it means:** If someone gives you a hashed value (say H(r ‖ x), where "‖" means joining two pieces of information together), it should be nearly impossible to figure out the original input (especially the hidden part, r).
* **Example:** Imagine you have a secret, and you add that secret to some other information before hashing it. Even if someone knows the other information, they shouldn't be able to figure out your secret just by looking at the hash.

**Min-entropy** is a concept that measures how unpredictable a value is. A high min-entropy means that the value is very random and hard to guess.

**1.1.3. Puzzle Friendliness**

* **What it means:** If you pick a random target hash value and try to find an input that produces that hash, it should be very difficult and take a long time.
* **Example:** Suppose you know a specific hash output (say "000...111"). Finding the exact input that produces this output should be so hard that it takes almost as long as guessing every possible combination.

This is like solving a puzzle where you know what the answer looks like, but it's nearly impossible to figure out the pieces that got you there.

**1.3. Why These Properties Matter**

In cryptography, these properties ensure that even if someone is trying to tamper with or steal your data, it's very difficult to do so. For example:

* **Collision resistance** helps make sure that no two different inputs look the same after hashing.
* **Hiding** ensures that someone can't reverse-engineer sensitive information from the hash.
* **Puzzle friendliness** prevents attackers from easily guessing inputs to match specific hash outputs.

These properties help protect against things like file tampering, ensuring data is safe during transmission or storage, and making it difficult for attackers to guess or manipulate secure information.

**1.4. Merkle-Damgård Transform**

* Hash functions need to handle inputs of any size. The **Merkle-Damgård transform** is a way to make this work.
* **How it works**: Instead of processing the entire input at once, the input is split into smaller blocks. These blocks are processed one by one, and each block’s output is fed into the next block’s input. For the first block, a special starting value called an **Initialization Vector (IV)** is used.
* This process continues until all blocks are processed, and the result from the last block becomes the final hash.

**1.5. Compression Function**

* **What it is:** A core part of the hash function that takes a fixed-size input and creates a smaller fixed-size output. In SHA-256, the compression function takes a 768-bit input and produces a 256-bit output.
* **Why it's important:** If this compression function is secure (meaning it's resistant to collisions), then the whole SHA-256 hash function will also be secure.

**1.6. Padding the Input**

* If the input isn't the perfect size for the block (like 512 bits in the case of SHA-256), extra bits are added to the input (called **padding**) to make it fit. This padding ensures that every block is exactly the right size.

**7. Random Oracle Model**

* In some cases, hash functions are modeled as random processes, meaning that for every input, the hash function behaves as if it randomly picks a new output. This is called the **random oracle model**.

**1.2**

A **hash pointer** is like a **normal pointer**, which tells you where some data is stored, but with one key difference—it also stores the **hash of that data**. A **blockchain** is a type of **linked list**, but instead of regular pointers, it uses **hash pointers**.

**Tamper-Evident Log**

* A **tamper-evident log** is like a list where you can keep adding new data at the end, but if someone tries to change anything earlier in the list, you'll be able to **detect** the tampering. This is exactly what a **blockchain** does.

**The Genesis Block**

* The **genesis block** is the very first block in the chain. Every blockchain starts from this block.
* By keeping the **hash pointer at the head** safe, you can ensure that **any tampering in any block** can be detected.

**How This Works in Practice:**

1. You keep adding new data to the log (new blocks).
2. Each block has a hash pointer that ensures the data hasn’t been changed.
3. If anyone tries to change data in the past, the mismatch will ripple through the entire chain.
4. As long as the **head hash pointer** is secure, any tampering will be noticed.

**1. What is a Merkle Tree?**

* A **Merkle tree** is a special type of **binary tree** that uses **hash pointers**.
* It’s used to organize and verify data efficiently, making it a very useful structure in blockchain systems like Bitcoin.

**Structure of a Merkle Tree:**

* At the **bottom** of the tree, we have a number of **data blocks**. These are called the **leaves**.
* Each leaf is hashed, and then **grouped in pairs** (two at a time).
* We create a new block that stores the **hashes of both leaves** in the pair.
* Then, we repeat this process, grouping the new blocks (with their hashes) into pairs and hashing them again.
* We keep doing this until we reach a single block at the **top**, called the **root**.

**2. Why Merkle Trees are Useful:**

* **Efficient Verification:** Merkle trees allow you to verify whether a particular block of data is part of the tree without checking the entire tree. You only need to check the path from that block to the root, which is much faster.

**3. Proof of Membership:**

* To **prove** that a block of data is part of the Merkle tree, you just need to show the **hashes** along the path from that block to the **root**.
* Since each block’s hash depends on the previous one, if a block is changed, its hash will no longer match the rest of the tree.

**4. Sorted Merkle Tree:**

* A **sorted Merkle tree** is a regular Merkle tree, but the blocks at the bottom are arranged in **a specific order** (like alphabetical or numerical order).
* This helps with proving **non-membership**.

**5. Proof of Non-Membership:**

* To **prove that a block is NOT in the tree**, a **sorted Merkle tree** can help.
* You show the **path to the block before** and the **path to the block after** the place where the missing block should be.
* If those two blocks are next to each other, you’ve proven that the block in question is not in the tree, because there’s no space between them for the missing block.

**Example:** Let’s say we have blocks for Alice, Bob, and Charlie, arranged alphabetically. You want to prove that there’s no block for Daniel.

* You would show the path to **Charlie** and **Bob**, proving that there’s no room between them where **Daniel** would fit.

Merkle trees make data verification much faster and secure, which is why they are widely used in systems like blockchains.

**1.3**

A **digital signature scheme** is a method that allows you to sign a message so that others can verify its authenticity. Think of it like signing a document, but in a secure digital form. The key difference is that digital signatures ensure that the message hasn’t been tampered with and proves the identity of the person who signed it.

**Here's how it works:**

1. **Key Generation**:
   * First, you need a pair of keys: a **private key (sk)** and a **public key (pk)**.
   * The private key is **kept secret** and is used to create the digital signature.
   * The public key is **shared with others** so that they can verify your signature.

The process to generate these keys looks like this:

(sk, pk) := generateKeys(keysize)

* + The **private key (sk)** is 256 bits.
  + The **public key (pk)** comes in two forms: **compressed** (257 bits) and **uncompressed** (512 bits).

1. **Signing a Message**:
   * To sign a message, you use your **private key (sk)** and the message itself.
   * The signing process creates a unique **signature** for that specific message using your private key:
2. sig := sign(sk, message)
   * The **message** and the **signature** are typically 256 bits and 512 bits, respectively.

The signature is like a mathematical "seal" on the message, proving that you (the owner of the private key) created it.

1. **Verifying the Signature**:
   * When someone receives your message, they can check if the signature is valid using your **public key (pk)**.
   * They do this with the following process:
2. isValid := verify(pk, message, sig)
   * If the result is true, the signature is valid. If it’s false, either the message has been tampered with or the signature is not authentic.

**Two Important Properties:**

1. **Valid signatures must verify**:
   * If someone signs a message with their private key, then the signature should pass the verification check using their public key:
2. verify(pk, message, sign(sk, message)) == true
3. **Existentially unforgeable**:
   * This means that no one should be able to forge a valid signature for a message unless they know the private key. So, even if someone knows the public key and sees several signatures, they can't create a valid signature on their own without the private key.

**Summary:**

* **Private Key (sk)**: A secret 256-bit key used to create the signature.
* **Public Key (pk)**: A key shared with others to verify the signature (compressed is 257 bits, uncompressed is 512 bits).
* **Message**: The data you want to sign (256 bits).
* **Signature**: A 512-bit value created using the private key and the message.

The digital signature ensures two things:

1. **Integrity**: The message hasn’t been altered.
2. **Authenticity**: The message came from the person who owns the private key.

This is how digital signatures work in systems like email encryption, blockchain transactions, and secure communications.

**1.5**

**Summary of GoofyCoin Rules:**

1. **Coin Creation**:
   * Goofy, the creator of the coin, can issue new coins by signing a statement that creates a new coin with a unique coin ID.
2. **Transferring Coins**:
   * Whoever owns a coin can transfer it to someone else by signing a statement like: "Pass this coin to X" (where X is the recipient's public key).
3. **Verification**:
   * Anyone can verify that a coin is legitimate by tracing back its history through a chain of hash pointers, all the way to its creation by Goofy, while verifying the signatures along the way.

**The Security Problem: Double-Spending**

GoofyCoin faces a **double-spending attack** problem, where a person can spend the same coin twice. Here's how:

* **Example**: Alice has a coin and sends a signed statement to Bob, passing the coin to him.
  + However, Alice also sends another signed statement to Chuck, passing the same coin to him.
  + Now both Bob and Chuck have valid signatures proving ownership of the same coin.

Since both claims look legitimate, it causes confusion and breaks the system's integrity. This is known as **double-spending**, and it’s a major problem that any cryptocurrency must address.

use if statement to check wheather this coin is been used.

**ScroogeCoin: Solving the Double-Spending Problem with Centralization**

**ScroogeCoin** is a cryptocurrency designed to solve the double-spending problem by introducing an append-only ledger controlled by a central figure named **Scrooge**. Here's how it works:

**Key Features of ScroogeCoin:**

1. **Append-Only Ledger**:
   * Scrooge maintains a ledger containing all past transactions.
   * The ledger's append-only nature ensures that once a transaction is recorded, it cannot be altered or removed.
2. **Blockchain**:
   * Scrooge builds a blockchain by signing a chain of blocks, each containing transactions. Every block includes a hash pointer to the previous block.
   * This structure ensures that if any past transaction is modified, the change will cascade through the entire chain, making tampering detectable.
3. **Transaction Validation**:
   * Only transactions recorded in the blockchain signed by Scrooge are valid.
   * Scrooge prevents double-spending by verifying that each coin is spent only once.
4. **Two Types of Transactions**:
   * **CreateCoins**: Scrooge can create new coins and assign them to owners.
   * **PayCoins**: Owners can transfer their coins by signing transactions, which Scrooge verifies and records on the blockchain.

**ScroogeCoin Transaction Rules:**

For a **PayCoins** transaction to be valid, four conditions must be met:

1. The consumed coins are valid (created in previous transactions).
2. The coins haven't been consumed before (no double-spending).
3. The total value of coins going into the transaction equals the total value coming out (only Scrooge can create new coins).
4. The transaction is signed by all coin owners involved.

**Flexibility in Coin Usage:**

While coins themselves are immutable (can't be subdivided or combined), users can achieve these operations through transactions. For instance, to subdivide a coin, a user can create a transaction that consumes one coin and produces two new ones of equal total value.

**The Problem with ScroogeCoin: Centralization**

ScroogeCoin successfully prevents double-spending, but it introduces a central authority—**Scrooge**—which has significant control over the system:

* Scrooge could refuse to process certain users' transactions, making their coins unspendable.
* He could demand transaction fees or create unlimited new coins for himself.
* Scrooge could abandon the system entirely, halting all transactions.

This **centralized control** over the cryptocurrency is undesirable. Users must trust Scrooge not to act selfishly or abandon the system, which leads to significant limitations in practice.

**The Need for Decentralization:**

To improve on ScroogeCoin, the goal is to **"descroogify"** the system, removing the central authority while still ensuring a tamper-proof, double-spend-resistant ledger. This would involve:

* A decentralized way to agree on a single shared blockchain history.
* A system where transactions and coin minting are handled without central control.

This decentralized approach is precisely what **Bitcoin** and other modern cryptocurrencies aim to achieve.

**Quiz**

**1. Merkle Tree and Proof of Membership**

* **Merkle Tree**: A Merkle tree organizes data by hashing each piece (called leaf nodes), then hashing pairs of hashes to form higher levels, and continuing this until a single root hash is created. It’s used to verify data integrity efficiently.
* **Proof of Membership**: To prove a specific piece of data is part of the Merkle tree, you only need the hashes leading from that data to the root hash. This way, you don’t reveal the entire dataset. This is called **Merkle Proof**.
* **Example**: If you want to verify "data1" is in a tree, you only need a few hashes instead of the whole tree, ensuring both privacy and verification.

**2. GoofyCoin and Double-Spending Attack**

* **GoofyCoin's Disadvantage**: GoofyCoin allows a user to spend the same coin twice (called **double-spending**) because it doesn’t have a secure consensus mechanism. There’s no decentralized system to confirm whether the coin has already been spent.
* **Double-Spending Attack**: A user can send the same coin to two different people, causing both to think they were paid.
* **Bitcoin's Solution**: Bitcoin uses a decentralized **Proof-of-Work** system where miners confirm transactions and agree on the blockchain’s state, preventing double-spending.

**3. Basic Steps of Consensus Mechanism in Blockchain**

* **Step 1**: Transactions are grouped into a block.
* **Step 2**: Nodes (miners or validators) validate the block to ensure transactions are correct (e.g., no double-spending).
* **Step 3**: Nodes agree (reach consensus) on which block is added to the chain.
* **Step 4**: The block is added to the blockchain, and it becomes part of the permanent ledger.
* **Key Concepts**: Block validation, agreement among nodes, and protection against malicious actors (using methods like Proof-of-Work or Proof-of-Stake).

**4. Rewards and Transaction Costs in Consensus Mechanism**

* **Rewards**: Miners or validators receive a reward (new coins) for securing the network by validating transactions and adding new blocks.
* **Transaction Fees**: Users pay transaction fees that go to miners. This incentivizes miners to include a user’s transaction in the next block.
* **Importance**: Rewards and fees keep the network secure and sustainable by motivating participants to continue validating transactions.

**5. Role of Cryptographic Hashing in Blockchain Security**

* **Hashing**: A cryptographic hash converts data into a unique string of characters. In blockchain, each block contains a hash that links it to the previous block.
* **Data Integrity**: Hashing ensures data hasn’t been altered. If someone tries to tamper with a block, the hash will change, breaking the chain.
* **Immutability**: Once a block is added, it’s virtually impossible to change without altering all subsequent blocks.
* **Example**: Hashing is like a digital fingerprint—if anything changes in a block, the fingerprint changes, alerting the network to tampering.

These simplified answers should help you remember the core concepts easily!

**Centralization:**

Centralization means that control or decision-making is concentrated in a single authority or a small group. For example, in a centralized system, like a bank, one central organization controls and manages all the transactions and data.

**Decentralization:**

Decentralization means that control or decision-making is spread out across many individuals or nodes, with no single entity in charge. In a decentralized system, like Bitcoin, no one person or organization controls the network; instead, it is managed collectively by many independent participants.

In short:

* **Centralization** = One place or group controls everything.
* **Decentralization** = Power is spread across many, with no single authority.

**Distributed Consensus Protocol in Bitcoin**

In a distributed consensus protocol, the goal is to have multiple nodes in a decentralized system agree on a common value, even if some of the nodes are faulty or malicious. For Bitcoin, this process must account for network imperfections (like latency) and deliberate attempts by malicious nodes to subvert the process.

**Properties of a Distributed Consensus Protocol:**

1. **Agreement**: All honest nodes must eventually agree on the same value.
2. **Validity**: The value agreed upon must have been generated by an honest node.

**Bitcoin's Consensus Algorithm (Simplified)**

The Bitcoin protocol is a type of **Proof-of-Work** consensus mechanism that operates in a decentralized manner. Here's a simplified version of how it works:

1. **Broadcasting New Transactions**:
   * When new transactions occur, they are broadcast to all nodes on the Bitcoin network.
2. **Nodes Collect Transactions into Blocks**:
   * Each node gathers new, valid transactions and groups them into a **block**.
3. **Random Node Broadcasting a Block**:
   * In each round, a random node (determined by solving a cryptographic puzzle through the Proof-of-Work mechanism) gets to broadcast its block. This randomness prevents any one node from monopolizing the consensus process.
4. **Other Nodes Validate the Block**:
   * Other nodes in the network validate the block by checking if the transactions in the block are legitimate, ensuring the transactions are unspent and have valid signatures. If the block is valid, nodes accept it.
5. **Nodes Express Agreement by Linking to the Block**:
   * Nodes signal their agreement by including the **hash** of the accepted block in the next block they attempt to create. This effectively builds on the accepted block, reinforcing its validity within the blockchain.

**Bitcoin Incentive Mechanisms**

Bitcoin uses two main incentive mechanisms to reward nodes (also called **miners**) that contribute to the consensus process:

1. **Block Reward**:
   * The miner who successfully creates a new block (solves the cryptographic puzzle) is rewarded with newly created bitcoins. This reward is part of a special transaction called the **coinbase transaction**, which is included in the block they create.
   * This block reward diminishes over time (halves approximately every four years) until all 21 million bitcoins have been mined.
2. **Transaction Fees**:
   * In addition to the block reward, miners also earn **transaction fees**.
   * When users create a Bitcoin transaction, they can set a transaction fee by making the total output value less than the total input value. This difference goes to the miner as an incentive to include that transaction in a block.
   * As the block reward decreases over time, transaction fees will become the primary incentive for miners.

**How Consensus Is Reached**

Bitcoin reaches consensus through a combination of these mechanisms:

* **Proof-of-Work**: Nodes compete to solve a computational puzzle (mining). The first one to solve it earns the right to propose a new block.
* **Longest Chain Rule**: Nodes always accept the longest valid chain of blocks as the correct history. This ensures that nodes eventually agree on the same version of the blockchain.
* **Economic Incentives**: Block rewards and transaction fees incentivize honest behavior because malicious actions would result in wasted effort and loss of potential rewards.

Together, these mechanisms enable Bitcoin to maintain decentralized consensus, even in the face of network imperfections and malicious attacks.

A malicious ISP alone cannot directly perform a double-spend attack without significant computational resources, typically through controlling or collaborating with miners. The computational effort would vary depending on the blockchain’s size, security, and mining power distribution, but for a well-established network like Bitcoin, it would require a vast amount of resources and effort to be successful.

why bit coins don't have identities?

idnetiti is hard in P2P system sybil attack

,pesudonominity is goal

Question 3:Let H be a hash function that is both hiding and puzzle‐friendly. Consider G(z) = H(z) ǁ z last where z last represents the last bit of z. Show that G is puzzle‐friendly but not hiding. [4 Points] Due to Last Bit Z is constant the given function is become predictable

Why do miners need to run "full nodes" that keep track of the entire blockchain, while normal users of blockchain-based applications can use "lite nodes" that implement "simplified payment verification" and only need to examine the last few blocks. [10 Points] ⎫ Minner Run full node inorder to check multiple security validation attribute to provide the security feature of all transaction such as double spending attack , 51% attack etc ⎫ Normal user or you say that transactional user of the network only require the header of the entire blockchain due use as a proof of membership purpose in order to process the transaction their fore it is not necessary to take the whole ledger of blockchain in normal user nodes

**4a. What determines whose block will end up on the consensus branch?**

When two miners, Minnie and Mynie, find blocks at nearly the same time, both will propagate their blocks through the network. However, only one of them will end up in the consensus chain. Here's how it gets decided:

* **Network Propagation**: The block that reaches a majority of miners first will likely be the one that gets accepted, as other miners will start building on top of it.
* **Longest Chain Rule**: Bitcoin’s consensus mechanism favors the longest valid chain (highest cumulative proof-of-work). As miners continue working, the chain that gets another block on top of it will become the longer chain. The shorter chain becomes "orphaned," and its block is discarded.

In summary, the block that gets a new block added first will stay in the consensus branch.

**4b. What factors affect the rate of orphan blocks? Can you derive a formula for the rate based on these parameters?**

Several factors influence the rate of orphan blocks:

1. **Block Propagation Time**: If blocks take longer to propagate across the network, there's a higher chance two miners will find blocks simultaneously.
2. **Block Time Interval**: Shorter intervals between blocks increase the chance of two miners finding blocks at the same time.
3. **Network Latency**: Higher latency in the network can lead to delayed block propagation, resulting in more orphaned blocks.
4. **Hashrate Distribution**: If mining power is more decentralized, there are more opportunities for multiple miners to discover blocks around the same time, leading to more orphan blocks.
5. **Block Size**: Larger blocks take longer to propagate, which can also increase the orphan rate.

While the exact formula is complex due to network dynamics, you can estimate the rate of orphan blocks (\( O \)) as a function of the **block time interval** (\( I \)) and **block propagation time** (\( P \)):

\[ O = f \left( \frac{P}{I} \right) \]

This formula shows that the orphan rate is proportional to the ratio of propagation time to the block interval. If propagation time \( P \) is a significant portion of \( I \), the orphan rate will increase.

**4c. Try to empirically measure this rate on the Bitcoin network.**

To measure the rate of orphan blocks on the Bitcoin network:

1. **Use Bitcoin Full Node**: Run a full Bitcoin node and keep track of the orphan blocks it receives.
2. **Blockchain Explorers**: Some blockchain explorers, like Blockchair, provide statistics on the number of orphan blocks.
3. **Data Analysis**: You could analyze block data over a certain period and check for blocks that were orphaned. [Blockchain.info](http://blockchain.info/) or Bitcoin's public APIs may also provide insights on recent orphan blocks.

Empirical analysis could be conducted by comparing the total number of orphan blocks to the total number of blocks created in a given timeframe to estimate the orphan rate.

**4d. If Mynie hears about Minnie’s block just before she’s about to discover hers, does that mean she wasted her effort?**

Yes, if Mynie hears about Minnie’s block right before she’s about to discover hers, her effort is generally considered wasted in terms of finding a valid block for the current round of mining. Here’s why:

* **Bitcoin’s Protocol**: Once Mynie knows about Minnie’s block, she will start mining on top of Minnie’s block instead of broadcasting her own, as her block would be invalid if it's built on an old version of the chain.
* **Effort Wasted**: The work Mynie did before hearing about Minnie’s block cannot be reused, so that computational effort is effectively wasted.

This phenomenon is called a "stale block."

**4e. Do all miners have their blocks orphaned at the same rate, or are some miners affected disproportionately?**

No, all miners are not equally affected. Some factors can cause certain miners to have a higher orphan block rate:

1. **Geographical Location**: Miners that are geographically farther from the majority of the network may experience higher latency, which can result in their blocks reaching other miners later, leading to a higher orphan rate.
2. **Mining Pool Size**: Larger mining pools have a lower orphan rate because they can propagate blocks faster due to their network size and infrastructure. Smaller, independent miners may have a higher orphan rate.
3. **Network Infrastructure**: Miners with faster internet connections and better infrastructure will generally have a lower orphan rate because they can propagate blocks more quickly.

In conclusion, miners with better connectivity, larger hash power, and lower latency are less likely to have their blocks orphaned compared to smaller or less well-connected miners.

the probability that a block will be found in the next 10 minutes is approximately **63.21%**.

100 minuts or higher