BT Unit 1 Notes By OMG

---

1. Differentiate between symmetric and asymmetric key cryptography.

Symmetric and asymmetric key cryptography are two fundamental methods of encryption, distinguished primarily by their use of keys.

- **Symmetric Key Cryptography** uses a **single, shared secret key** for both the encryption and decryption processes. For two parties to communicate securely, they must both have a copy of this same key and must keep it confidential. It is generally faster and less computationally intensive than asymmetric cryptography.
    - **Analogy:** Think of a physical lock and key. Anyone who has a copy of the key can both lock (encrypt) and unlock (decrypt) the box. The main challenge is securely giving a copy of the key to the intended recipient.
    - **Examples:** AES (Advanced Encryption Standard), DES (Data Encryption Standard).
- **Asymmetric Key Cryptography** (also known as public-key cryptography) uses a **pair of keys** for communication: a **public key** and a **private key**. The public key can be shared openly with anyone, while the private key must be kept secret by its owner. Data encrypted with the public key can *only* be decrypted with the corresponding private key.
    - **Analogy:** Imagine a mailbox with two keys. One key (the public key) can only lock the mailbox and can be widely distributed. Anyone can put a message in and lock it. However, only the person with the second, unique key (the private key) can open the mailbox and read the messages.
    - **Examples:** RSA (Rivest-Shamir-Adleman), Elliptic Curve Cryptography (ECC).

Here is a summary of the key differences:

Feature	Symmetric Cryptography	Asymmetric Cryptography
Number of Keys	One shared secret key	A pair of keys (public and private)
Key Sharing	The shared key must be distributed securely.	The public key can be shared openly.
Speed	Fast	Slow

Computational Load	Less intensive	More intensive
Primary Use Cases	Encrypting large amounts of data (data at rest, data in transit).	Secure key exchange, digital signatures, and authentication.

---

2. What is hashing? Explain role of hashing in Blockchain.

Hashing is a process that converts an input of any length (a message, file, or data) into a fixed-length string of characters, which is typically a hexadecimal number. This output is called a **hash**, hash value, or message digest. The function that performs this conversion is called a hash function. [11]

Key properties of cryptographic hash functions include:

- **Deterministic:** The same input will always produce the same output.
- **One-Way Function:** It is computationally infeasible to reverse the process and find the original input from its hash.
- **Fixed-Size Output:** Any input, regardless of its size, produces an output of a fixed length (e.g., SHA-256 always produces a 256-bit hash).
- **Collision Resistance:** It is extremely difficult to find two different inputs that produce the same hash output.

Role of Hashing in Blockchain

Hashing is a cornerstone of blockchain technology, essential for its security, immutability, and integrity. [222]

1. **Maintaining Data Integrity and Immutability:** Every block in a blockchain contains a hash of its own header, which includes the timestamp, transaction data (via a Merkle Root), and the hash of the previous block. If a single character in any transaction within a block is altered, the hash of that block changes completely. This, in turn, changes the hash of every subsequent block in the chain, immediately signaling that a tamper has occurred.
2. **Linking Blocks:** The "chain" in "blockchain" is created by hashing. Each block's header

contains the hash of the preceding block. This cryptographic link ensures that the history of the ledger is preserved in the correct order and cannot be altered without invalidating the entire chain that follows.

3. **Proof-of-Work (PoW) Consensus:** In blockchains like Bitcoin, miners compete to add the next block to the chain. They do this by repeatedly hashing the block's header along with a random number called a "nonce." The goal is to find a hash that is below a certain target value (i.e., starts with a specific number of zeros). This process, known as mining, is computationally intensive and secures the network. Hashing is the central operation in this process.
4. **Transaction Summarization:** Hashing is used to create a Merkle Tree (discussed in the next question), which efficiently summarizes all transactions in a block into a single hash value called the Merkle Root.

---

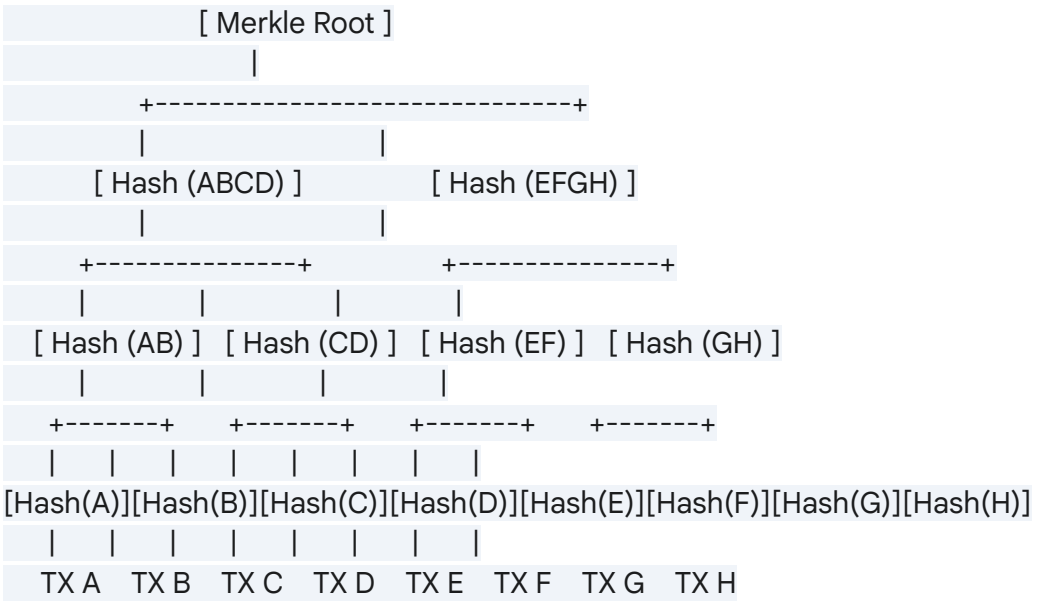
3. What is Merkle tree? Explain with diagram.

A **Merkle tree**, also known as a hash tree, is a data structure in which every leaf node is a hash of a block of data, and every non-leaf node is a hash of its child nodes. It's a way to efficiently and securely verify the contents of a large data structure. [33] In the context of a blockchain, the "data" is the set of transactions within a block.

How a Merkle Tree is Constructed:

1. **Hashing Transactions:** Each individual transaction (TX) in a block is hashed. These hashes form the leaf nodes of the tree.
2. **Pairing and Hashing:** The leaf nodes are grouped into pairs, and the hash of each pair is calculated. For example, Hash(TX A) and Hash(TX B) are concatenated and hashed together to form a parent hash, Hash(AB).
3. **Repeating the Process:** This process is repeated up the tree. The parent hashes are paired and hashed again, creating another level of the tree.
4. **The Merkle Root:** The process continues until only a single hash remains. This final hash is called the **Merkle Root** or the root hash. The Merkle Root is stored in the block header and serves as a summary of all transactions in that block.

Diagram Explained:



- At the bottom are the individual transactions (TX A, TX B, etc.).
- Each transaction is hashed.
- Pairs of transaction hashes are combined and hashed to create parent nodes (e.g., Hash(A) and Hash(B) create Hash(AB)).
- This continues until the single Merkle Root is produced at the top.

---

4. Explain digital signature algorithm.

A **Digital Signature Algorithm (DSA)** is a mathematical scheme for verifying the authenticity of digital messages or documents. Based on asymmetric cryptography, a valid digital signature gives a recipient strong reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity). [444]

The process involves three main stages:

1. **Key Generation:**
    - An individual (let's call her Alice) generates a key pair consisting of a **private key** and

a corresponding **public key**.
    - The private key is kept secret and is used for creating the signature.
    - The public key is made available to anyone who needs to verify Alice's signature.
2. **Signature Generation (Signing):**
    - Alice takes the original message she wants to send.
    - She uses a hash function to produce a fixed-size message digest (hash) of the message.
    - Alice then uses her **private key** to encrypt this hash. The encrypted hash is the **digital signature**.
    - Finally, she sends the original message along with the digital signature to the recipient (Bob).
3. **Signature Verification:**
    - Bob receives the message and the signature.
    - He uses the same hash function Alice used to generate a hash of the received message.
    - Bob then uses Alice's **public key** to decrypt the digital signature. This reveals the original hash that Alice created.
    - Bob compares the hash he calculated from the message with the decrypted hash from the signature. If the two hashes match, the signature is valid. This proves the message came from Alice and was not altered.

---

5. What are the benefits of Merkle tree in Blockchain.

Merkle trees provide several significant benefits in blockchain technology, primarily related to efficiency and security. [5]

1. **Efficient Data Verification (Simple Payment Verification - SPV):** The most important benefit is that Merkle trees allow users to verify if a transaction is included in a block without downloading the entire block's data. A "light node" client only needs the block headers. To verify a transaction, it can request the Merkle Root from the header and the "Merkle path" or "branch" (the series of hashes connecting the transaction to the root). By recomputing the hashes up this path, the client can confirm that the transaction is part of the Merkle Root, and thus part of the block, using a very small amount of data.
2. **Data Integrity:** The Merkle Root provides a single, fixed-size hash that summarizes all the transactions within a block. If even a single detail of a single transaction is changed, its hash will change. This change will cascade up the Merkle tree, ultimately resulting in a different Merkle Root. Since the Merkle Root is included in the block hash, any tampering with transactions becomes immediately evident, as it would invalidate the block and break the entire chain.
3. **Reduced Data Transfer and Storage:** For verification purposes, only the Merkle path is

needed, not the full set of transactions. This drastically reduces the amount of data that needs to be transferred over the network and stored by light clients, making the blockchain more scalable and accessible to devices with limited resources.

---

6. Discuss elliptic curve cryptography.

Elliptic Curve Cryptography (ECC) is a modern approach to public-key cryptography based on the mathematical properties of elliptic curves. It is used to create faster, smaller, and more efficient cryptographic keys compared to older methods like RSA, while providing an equivalent level of security. [6]

How ECC Works:

An elliptic curve is a specific type of mathematical curve defined by an equation like $y^2=x^3+ax+b$. In ECC, this curve is defined over a finite field, meaning the points on the curve are limited to a fixed range of integer values.

The core of ECC's security lies in the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**. Here's a simplified breakdown of the key generation process:

1. A standard elliptic curve and a "generator point" (G) on that curve are publicly agreed upon.
2. **Private Key:** A user (Alice) generates a private key, which is simply a large, randomly selected integer (d). This key is kept secret.
3. **Public Key:** Alice computes her public key (Q) by adding the generator point G to itself d times ($Q = d * G$). This operation is easy to perform in one direction.
4. **The Hard Problem:** However, it is computationally infeasible for anyone to determine the private key d even if they know the public key Q and the generator point G. This one-way nature makes ECC a secure foundation for cryptography.

Advantages of ECC in Blockchain:

- **Smaller Key Sizes:** For the same level of security, ECC keys are significantly shorter than RSA keys. For example, a 256-bit ECC key provides comparable security to a 3072-bit

RSA key.

- **Increased Efficiency:** The smaller key sizes lead to faster computations, lower power consumption, and less storage and bandwidth requirements. This is particularly beneficial for resource-constrained environments like mobile devices and for technologies like blockchain that need to process and store a massive number of signatures. Bitcoin and Ethereum, for instance, use ECC for generating public-private key pairs and for creating digital signatures.

---

7. Justify the importance of Hashing in Block Chain.

Hashing is not just important to blockchain; it is the fundamental technology that makes blockchain's core properties possible. Its importance can be justified by its role in enabling immutability, security, and the decentralized consensus mechanism. [7]

1. **Foundation of Immutability:** The defining feature of a blockchain is its immutable ledger. This is achieved directly through hashing. Each block is cryptographically linked to the previous one by including the parent block's hash in its own header. Altering data in any past block would change that block's hash, causing a mismatch in the next block, and every subsequent block thereafter. This instantly breaks the chain, making any tampering immediately obvious. Without hashing, this unchangeable, chained structure could not exist.
2. **Guarantee of Data Integrity:** Hashing guarantees the integrity of the data within each block. All transactions in a block are summarized into a single Merkle Root hash, which is included in the block header. This ensures that no transaction can be altered or removed without changing the Merkle Root, which in turn would change the block hash and break the chain. Hashing acts as a tamper-proof seal for the block's content.
3. **Enabler of Consensus and Security (Proof-of-Work):** In many major blockchains, the security and decentralized nature are maintained through Proof-of-Work (PoW). PoW is essentially a computationally difficult hashing puzzle. Miners must repeatedly hash a block's header until they find a hash that meets a certain difficulty criterion. This process is vital because:
    - It makes creating new blocks costly and time-consuming, preventing malicious actors from easily adding fraudulent blocks.
    - It serves as the mechanism for achieving distributed consensus on the state of the ledger in a trustless environment.
      The entire PoW model, which secures billions of dollars in value, is built upon the properties of cryptographic hash functions.

In essence, hashing provides the mechanism for creating digital fingerprints for data, linking

blocks into a secure chain, and running the consensus engine that governs the network.

---

8. Explain working of SHA 256 Algorithm.

SHA-256 (Secure Hash Algorithm 256-bit) is a specific cryptographic hash function from the SHA-2 family, which is widely used in blockchain (e.g., Bitcoin) and other security protocols like TLS. It takes any input and produces a 256-bit (32-byte) hash output. [8]

The algorithm works in several key steps:

1. **Preprocessing (Padding and Appending Length):**
   - The input message is padded so that its total length in bits is just 64 bits shy of a multiple of 512. Padding starts by appending a '1' bit, followed by the necessary number of '0' bits.
   - The original length of the message (in bits) is then calculated and appended to the end as a 64-bit integer. The result is a message that is a precise multiple of 512 bits in length.
2. **Initialization of Hash Buffers:**
   - SHA-256 maintains an internal state using eight 32-bit variables (or buffers), labeled h0 through h7.
   - These are initialized with constant values derived from the fractional parts of the square roots of the first eight prime numbers (2, 3, 5, 7, 11, 13, 17, 19).
3. **Processing the Message in 512-bit Chunks:**
   - The padded message is divided into 512-bit chunks. The core of the algorithm, called the **compression function**, is applied to each chunk in sequence.
   - For each chunk, the eight hash buffers are copied into eight working variables (a, b, c, d, e, f, g, h).
4. **The Compression Function Loop:**
   - This is the main loop where the data is processed. It runs for 64 rounds for each 512-bit chunk.
   - In each round, the values of the working variables are mixed and transformed using a series of complex bitwise operations, including bitwise AND, OR, XOR, NOT, circular shifts (rotate), and addition.
   - These operations involve the working variables, pre-defined constants (k[t]), and a "message schedule" (W[t]) that is derived from the current 512-bit message chunk.
   - The result of these 64 rounds is used to update the values in the main hash buffers (h0 through h7).
5. **Final Hash Value:**
   - After the final message chunk has been processed through the compression

function, the final values in the eight 32-bit hash buffers (h0 to h7) are concatenated.
    - This concatenated 256-bit value is the final hash digest of the original message.

---

9. Describe Digital Signature & Verification steps in Digital Signature Algorithm.

Here is a step-by-step description of the signing and verification processes within a Digital Signature Algorithm (DSA). [9]

Digital Signature Generation Steps

This process is performed by the sender (e.g., Alice) who wants to prove the authenticity and integrity of her message.

1. **Create a Message Digest:** Alice takes her original message (M) and applies a cryptographic hash function (like SHA-256) to it. This produces a fixed-size hash value, H(M).
2. **Generate Signature Components:** Alice uses her **private key** along with the hash H(M) as inputs to the signing algorithm. The algorithm outputs two values, typically named r and s, which together form the digital signature.
3. **Send the Data:** Alice sends the original message (M) and the digital signature (r, s) to the recipient (e.g., Bob).

Digital Signature Verification Steps

This process is performed by the recipient (Bob) to confirm that the message truly came from Alice and was not altered.

1. **Obtain Sender's Public Key:** Bob must have a trusted copy of Alice's **public key**. This is crucial for the verification to be secure.
2. **Create a Message Digest:** Bob takes the received message (M) and applies the *exact same* hash function that Alice used. This produces a hash value, H(M).
3. **Run the Verification Algorithm:** Bob feeds Alice's public key, the received signature (r, s), and the hash he just computed (H(M)) into the verification algorithm.

4. **Compare the Output:** The verification algorithm produces a value. This output value is then compared to the r value from the signature.
    - **If they match,** the signature is **valid**. This mathematically proves two things:
        - **Authenticity:** The signature was created using the private key corresponding to Alice's public key, so it must have been Alice who signed it.
        - **Integrity:** The message has not been changed since it was signed, because its hash still matches the one embedded in the signature.
    - **If they do not match,** the signature is **invalid**. This indicates that either the message was tampered with or it was not signed by Alice.

---

10. Describe Symmetric Key Encryption with neat diagram.

Symmetric Key Encryption is an encryption method where a **single key** is used for both encrypting plaintext into ciphertext and decrypting ciphertext back into plaintext. This key is often referred to as a "shared secret" because both the sender and the receiver must possess it and keep it confidential from all other parties. [10]

It is a fast and efficient method for encrypting large volumes of data. Its main operational challenge is **key distribution**—the problem of securely sharing the secret key with the intended recipient without it being intercepted.

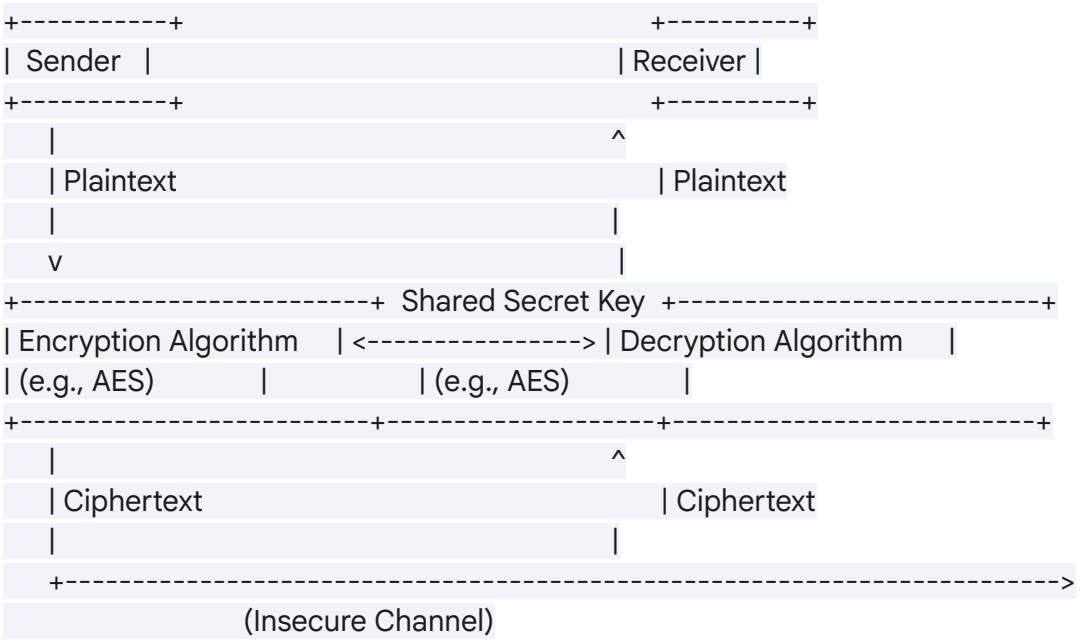
The Process:

- **Encryption:** The sender combines the plaintext message with the shared secret key using an encryption algorithm (like AES) to produce the ciphertext.
    - Ciphertext = Encrypt(Plaintext, Shared Key)
- **Decryption:** The receiver uses the exact same shared secret key with a decryption algorithm to convert the ciphertext back into the original plaintext.
    - Plaintext = Decrypt(Ciphertext, Shared Key)

If an unauthorized person intercepts the ciphertext but does not have the shared key, the message remains unreadable.

Diagram of Symmetric Key Encryption:

The flow can be visualized as follows:



- The **Sender** uses the encryption algorithm and the **Shared Secret Key** to turn the **Plaintext** into **Ciphertext**.
- The **Ciphertext** is sent over an insecure channel (like the internet).
- The **Receiver** uses the same **Shared Secret Key** and the decryption algorithm to turn the **Ciphertext** back into the original **Plaintext**.