UNIT-1

c) Describe Symmetric Key Encryption with neat diagram. [5]

Symmetric-Key Encryption: Exam Notes

What is Symmetric-Key Encryption?

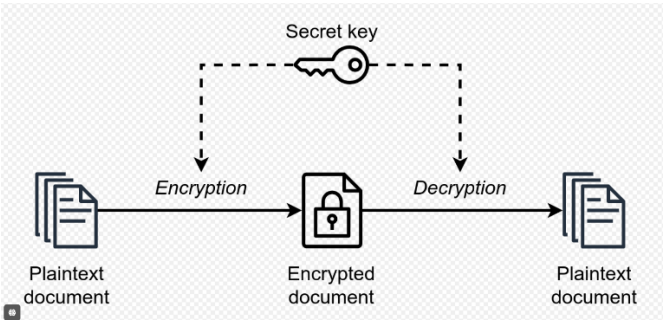
Symmetric-key encryption is a method of encrypting data where the same secret key is used to both encrypt and decrypt information. Both the sender and the receiver need to have this secret key to read the encrypted data.

How It Works

1. **Encryption**: The sender uses the secret key to turn readable information (plaintext) into unreadable information (ciphertext).
2. **Transmission**: The encrypted data (ciphertext) is sent to the recipient.
3. **Decryption**: The recipient uses the same secret key to turn the unreadable data (ciphertext) back into readable information (plaintext).

Diagram of Symmetric-Key Encryption

Here's a simple diagram to show how symmetric-key encryption works:



Types of Symmetric-Key Encryption

1. **Block Ciphers**: Encrypt data in chunks (blocks) of fixed size. Example: AES (Advanced Encryption Standard).
2. **Stream Ciphers**: Encrypt data one bit or byte at a time. Example: RC4.

Advantages of Symmetric-Key Encryption

1. **Speed**: Symmetric-key encryption is usually faster and more efficient, making it good for encrypting large amounts of data.
2. **Simplicity**: The method and the key management are easier compared to other encryption methods.

Disadvantages of Symmetric-Key Encryption

1. **Key Sharing**: Both parties need to securely share and keep the same key secret. This can be tricky and risky.
2. **Scalability**: In a network with many users, managing many keys becomes complex.

Example Algorithms

- **AES (Advanced Encryption Standard)**: A popular and secure method used today.
- **DES (Data Encryption Standard)**: An older method that is now less secure but still used in some cases.

Symmetric-key encryption is great for protecting data when speed and simplicity are important, but it requires careful management of the secret key to keep data safe.

What is Asymmetric Encryption?

Asymmetric encryption, also known as public-key cryptography, uses two different keys for encryption and decryption:

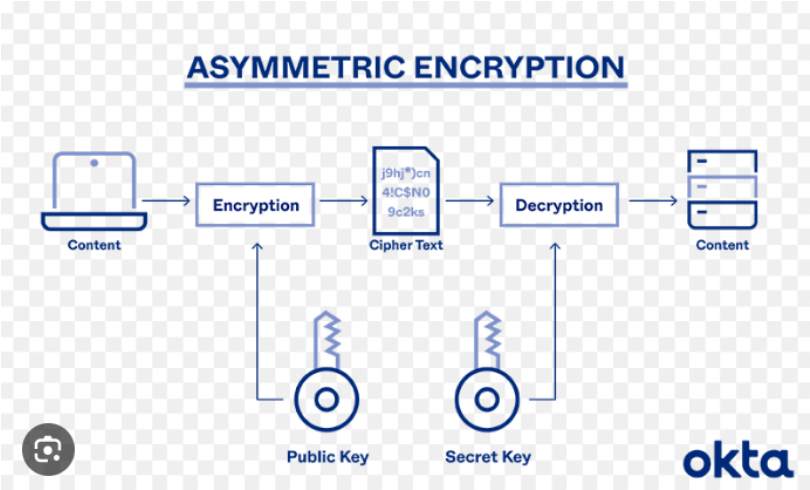
- **Public Key**: This key can be shared with anyone. It is used to encrypt data.
- **Private Key**: This key is kept secret by the owner. It is used to decrypt data.

How It Works

1. **Encryption**: The sender uses the recipient's public key to encrypt the message.
2. **Transmission**: The encrypted message is sent to the recipient.
3. **Decryption**: The recipient uses their private key to decrypt the message and read it.

Diagram of Asymmetric Encryption

Here's a simple diagram to illustrate how asymmetric encryption works:



Benefits of Asymmetric Encryption

1. **Better Security**: It's more secure because it uses two different keys. The public key is shared openly, while the private key stays secret.
2. **Verification:** You can check who sent the message. If a message is locked with a sender's private key, it means only the sender could have sent it.
3. **No Key Sharing Needed**: You don't need to share a secret key with the recipient. The public key is shared freely, and only the private key is kept secret.
4. **Useful in Many Ways**: It's used in secure emails, online banking, and digital signatures to keep data safe.

Examples of Asymmetric Encryption

- **RSA**: Often used for sending data securely.
- **Diffie-Hellman**: Used for securely sharing keys.
- **Elliptic Curve Cryptography (ECC)**: Provides strong security with shorter keys.

How Asymmetric Encryption is Used in TLS/SSL

1. **TLS Handshake**: Asymmetric encryption is used to safely exchange keys that will be used for the actual data transfer.
2. **Session Keys**: Once keys are exchanged, symmetric encryption is used to quickly handle the data during the session.

Conclusion

Asymmetric encryption is a key method for securing data. It uses two keys to lock and unlock information, making it safer and easier to manage. This method is widely used to keep communications secure.

b) Differentiate asymmetric and symmetric key cryptography. **[5]**

Symmetric Key Encryption

Symmetric Key Encryption is a method where the same key is used to lock (encrypt) and unlock (decrypt) a message. It's fast and good for encrypting large amounts of data. However, the main challenge is securely sharing the key between the sender and receiver.

Asymmetric Key Encryption

Asymmetric Key Encryption uses two different keys: a public key to lock (encrypt) the message and a private key to unlock (decrypt) it. This method is more secure because the keys are different, but it is slower than symmetric encryption and is typically used for smaller amounts of data.

Symmetric Key Encryption	Asymmetric Key Encryption
Uses a single key for both encryption and decryption.	Uses two keys: one for encryption (public) and anothe for decryption (private).
The same key must be shared between sender and receiver.	Only the public key is shared; the private key remains secret.
The encryption process is very fast.	The encryption process is slower.
Ideal for large data transfers.	Ideal for small data transfers.
Provides confidentiality only.	Provides confidentiality, authenticity, and non-repudiation.
Key length is typically 128 or 256 bits.	Key length is typically 2048 bits or more.
Lower security as one key is used for both tasks.	Higher security as different keys are used.
Less resource-intensive; requires fewer computing resources.	More resource-intensive; requires more computing resources.
Easier to implement and manage.	More complex to implement and manage.
Vulnerable if the key is intercepted during transfer.	Secure even if the public key is known by others.
The size of the ciphertext is the same or smaller thar the plaintext.	The size of the ciphertext is often larger than the plaintext.
Commonly used in applications like data encryption and secure communication.	Commonly used for secure data transmission and digital signatures.
Examples: AES, DES, 3DES, RC4.	Examples: RSA, ECC, Diffie-Hellman, DSA.

*

Exam Preparation Notes: Elliptic Curve Cryptography (ECC)

1. Introduction to Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a type of public-key encryption that uses the mathematical properties of elliptic curves over finite fields. ECC provides similar security to other public-key encryption methods, like RSA, but with much shorter key lengths. This makes ECC more efficient in terms of computational power, bandwidth, and storage.

2. Key Concepts and Components of ECC

- **Elliptic Curves:** Defined by the equation y2=x3+ax+by^2 = x^3 + ax + by2=x3+ax+b over a finite field, where `a` and `b` are constants. The points on this curve represent possible keys.
- **Private Key:** A random integer within a specified range. It is kept secret.
- **Public Key:** A point on the elliptic curve, generated by multiplying the private key with a generator point on the curve.
- **Generator Point (G):** A predefined point on the curve that is used to generate public keys.
- **Field Size:** Determines the range of values for `x` and `y` coordinates on the elliptic curve. Larger fields provide more security.

3. ECC Algorithms

- **Elliptic Curve Digital Signature Algorithm (ECDSA):** Used for digital signatures. It is more efficient than RSA and provides the same security with shorter keys.
- **Elliptic Curve Diffie-Hellman (ECDH):** A key exchange protocol that allows two parties to securely share a secret key over an insecure channel.
- **Elliptic Curve Integrated Encryption Scheme (ECIES):** Combines ECC with symmetric encryption for secure data transmission.

4. Applications of ECC

- **Cryptocurrencies:** ECC is widely used in cryptocurrencies like Bitcoin and Ethereum for secure transactions.
- **Secure Communications:** ECC is used in HTTPS and SSL/TLS protocols for secure internet communications.
- **Digital Signatures:** ECDSA, an ECC variant, is used in digital signatures for verifying the authenticity of messages and transactions.

5. Advantages of ECC

- **Smaller Key Size:** ECC provides equivalent security to RSA with much smaller keys (e.g., a 256-bit ECC key is as secure as a 3072-bit RSA key).
- **Faster Computations:** Shorter keys result in faster encryption and decryption processes, which is critical for devices with limited processing power like smartphones.
- **Lower Bandwidth:** Smaller keys and faster computations mean less data is transmitted, saving bandwidth.

- **High Security:** Despite shorter key lengths, ECC is highly secure due to the complexity of the elliptic curve discrete logarithm problem.

6. Limitations of ECC

- **Complex Implementation:** ECC is mathematically complex, making it harder to implement correctly.
- **Vulnerability to Quantum Computing:** Like other public-key systems, ECC could be vulnerable to quantum attacks in the future.

Question: Explain the advantages of using Elliptic Curve Cryptography (ECC) over RSA in secure communications. Include a discussion on key size and computational efficiency.

Answer: Elliptic Curve Cryptography (ECC) is preferred over RSA in secure communications due to its ability to provide equivalent security with significantly smaller key sizes. For instance, while a 3072-bit RSA key is necessary to achieve 128-bit security, ECC can achieve the same level of security with just a 256-bit key. This smaller key size results in faster encryption and decryption processes, making ECC more efficient, particularly for devices with limited computational power, such as smartphones and IoT devices.

Additionally, the reduced key size in ECC also translates to lower bandwidth requirements, which is advantageous in network communications where bandwidth is a critical resource. Despite the efficiency, ECC maintains a high level of security due to the difficulty of solving the elliptic curve discrete logarithm problem, making it a robust choice for modern secure communications.

Q1) a) List and explain advantages of ECC. **[5]**

Question: List and explain advantages of ECC (Elliptic Curve Cryptography).

Answer:

Elliptic Curve Cryptography (ECC) offers several advantages in cryptography:

1. **Stronger Security with Smaller Keys**:

ECC provides a high level of security with smaller key sizes compared to other methods like RSA. For example, a 256-bit ECC key provides similar security to a 3072-bit RSA key. This makes ECC efficient and secure.

2. **Faster Computation**:

Because of the smaller key sizes, ECC requires less processing power. This means encryption and decryption can be done faster, which is especially useful for devices with limited computational resources, like smartphones.

3. **Less Memory Usage**:

Smaller keys and faster computation lead to lower memory requirements. ECC uses less storage and bandwidth, making it ideal for systems where memory and data transfer capabilities are limited.

4. **Energy Efficiency**:

ECC consumes less energy due to its efficiency in processing. This is particularly beneficial for battery-powered devices, such as mobile phones or IoT (Internet of Things) devices, where power consumption needs to be minimized.

5. **Scalability**:

ECC can be easily scaled up to higher levels of security without significantly increasing the key size. This ensures that ECC can adapt to future security needs without much overhead.

These advantages make ECC a preferred choice in modern cryptography, especially in environments where resources are limited but strong security is necessary.

Q1) a) Illustrate Elliptic Curve Gyptography. [6]

Illustrate Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a public-key cryptosystem that provides high security with smaller key sizes compared to other cryptosystems like RSA. ECC is based on the mathematics of elliptic curves, which are equations of the form:

y2=x3+ax+by^2 = x^3 + ax + by2=x3+ax+b

where 4a3+27b2≠04a^3 + 27b^2 \neq 04a3+27b2□=0 to ensure the curve has no singular points.

1. Basic Concepts:

- **Elliptic Curve:** An elliptic curve is a set of points that satisfy the equation y2=x3+ax+by^2 = x^3 + ax + by2=x3+ax+b over a finite field. The points on the curve, together with a special point at infinity, form an abelian group under an operation called point addition.
- **Point Addition:** Given two points PPP and QQQ on the curve, their sum R=P+QR = P + QR=P+Q is also a point on the curve. The rules for point addition are geometric in nature.
- **Scalar Multiplication:** In ECC, scalar multiplication refers to adding a point PPP to itself kkk times to get another point QQQ on the curve. This operation is denoted as Q=kPQ = kPQ=kP and forms the basis for ECC encryption and decryption.

2. ECC Key Pair Generation:

- **Private Key (k):** A randomly chosen integer within a certain range.
- **Public Key (Q):** Computed as Q=kPQ = kPQ=kP, where PPP is a predefined point on the curve known as the base point.

3. ECC Encryption and Decryption:

- **Encryption:**
    1. The sender knows the recipient's public key QQQ.

2. The sender selects a random integer rrr and computes two values:
    - C1=rPC_1 = rPC1=rP
    - C2=M+rQC_2 = M + rQC2=M+rQ
3. The ciphertext is the pair (C1,C2)(C_1, C_2)(C1,C2).

- **Decryption:**
    1. The recipient uses their private key kkk to compute rQrQrQ from C1C_1C1.
    2. The recipient subtracts rQrQrQ from C2C_2C2 to recover the original message MMM.

4. Advantages of ECC:

- **Smaller Key Size:** ECC offers comparable security to traditional systems like RSA but with much smaller key sizes (e.g., a 256-bit key in ECC is equivalent to a 3072-bit key in RSA).
- **Efficiency:** Smaller key sizes result in faster computations and reduced storage requirements, making ECC suitable for devices with limited resources.

5. Applications of ECC:

- **Digital Signatures:** ECC is widely used in digital signatures, particularly in protocols like ECDSA (Elliptic Curve Digital Signature Algorithm).
- **Secure Communications:** ECC is employed in secure communication protocols like SSL/TLS to establish secure connections.

Conclusion:

ECC is a powerful cryptographic technique that offers high security with efficient performance, making it ideal for modern cryptographic needs, especially in environments where resources are constrained.

c) What is Merkle tree? Explain the structure of merkle tree. **[5]**

c) What is Merkle tree? Explain the structure of merkle tree. **[5]**

Merkle Tree: Exam Notes

What is a Merkle Tree?

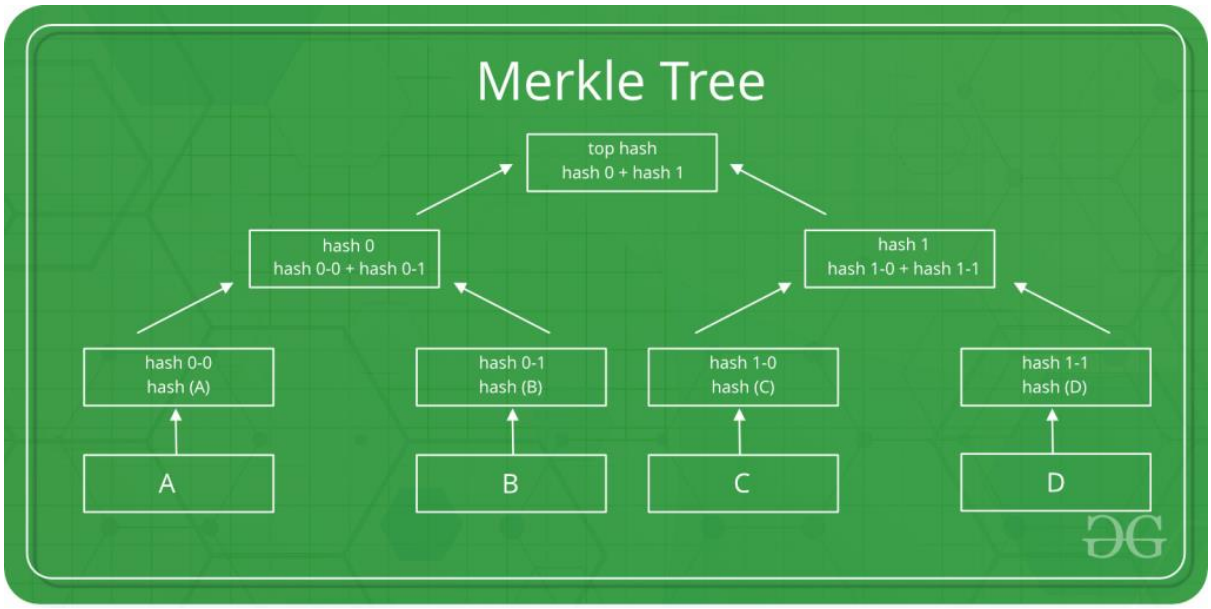
A Merkle Tree, also known as a hash tree, is a type of data structure that is used to verify and synchronize data. It is widely used in blockchain technology, like Bitcoin, to ensure data integrity. The tree consists of a series of hashes, with each non-leaf node containing the hash of its child nodes. The leaf nodes represent the data, while the non-leaf nodes are used to verify and manage the data.

Structure of a Merkle Tree

- **Leaf Nodes**: These are the bottom-most nodes in the tree, containing the actual data. Each piece of data is hashed to create a unique fingerprint.

- **Non-Leaf Nodes**: These nodes are higher up in the tree and each contains a hash derived from its child nodes.

- **Root Hash**: This is the topmost node of the tree and represents the entire dataset. It is the hash of the two child nodes directly beneath it. The root hash is used as a unique identifier for the entire dataset.



In a binary Merkle tree, every pair of nodes is hashed together until a single hash remains at the top of the tree, known as the root hash. If the number of leaf nodes is odd, one of them is duplicated to make it even.

Benefits of Merkle Tree

1. **Data Integrity**: Merkle trees ensure that any small change in data will change the hash, making it easy to detect data tampering.

2. **Efficient Verification**: You can verify data integrity without having to check the entire dataset. By comparing the root hash, you can determine if the data has been altered.

3. **Synchronization**: Merkle trees help in syncing data across different locations or nodes efficiently. Only parts of the tree that have changed need to be updated.

4. **Space Efficiency**: Merkle trees are space-efficient as they only store hashes, which are fixed in size, regardless of the data size.

5. **Security**: Hash functions used in Merkle trees are designed to be irreversible and unique, providing a high level of security.

Explanation of How Merkle Trees Work

- **Step 1**: Each piece of data (leaf node) is hashed individually.

- **Step 2**: These hashes are then paired and hashed together to form the next level of non-leaf nodes.

- **Step 3**: This process repeats until only one hash remains, the root hash, which represents the entire dataset.

- **Step 4**: To verify data integrity, the root hash is compared. If it matches the expected root hash, the data is considered intact.

Merkle trees are useful in distributed systems like blockchain, where the same data needs to exist in multiple places. They allow quick verification of data integrity and make synchronization more efficient.

CRYPTOGRAPHIC HASH FUNCTIONS:

Cryptographic Hash Function - Exam Notes

Definition:

- A cryptographic hash function is a mathematical algorithm that takes an input (or "message") and returns a fixed-size string of bytes. The output is typically a "digest" that is unique to each unique input.

Key Characteristics:

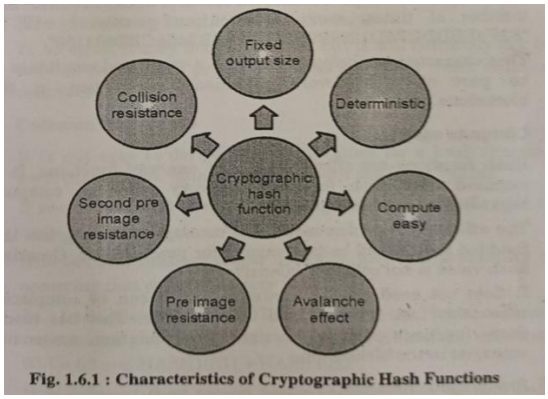


Fig. 1.6.1 : Characteristics of Cryptographic Hash Functions

1. **Deterministic**: The same input will always produce the same output (hash).

2. **Fast Computation**: Hash functions should be able to process data quickly.

3. **Pre-image Resistance**: It should be difficult to reverse-engineer the original input from the output hash.

4. **Small Changes in Input Produce Large Changes in Output**: Even a tiny change in the input should drastically change the output hash (Avalanche Effect).

5. **Collision Resistance**: It should be hard to find two different inputs that produce the same hash output.

Examples of Cryptographic Hash Functions:

1. **SHA-256 (Secure Hash Algorithm 256-bit)**

- **What It Is**: SHA-256 is a widely used cryptographic hash function that produces a 256-bit (32-byte) hash value. It is part of the SHA-2 family of hash functions.

- **Usage**: Commonly used in blockchain technology (e.g., Bitcoin), digital signatures, and certificate verification.

- **Security**: SHA-256 is considered very secure and is resistant to collision attacks, making it a reliable choice for cryptographic applications.

2. **Digital Signatures**

- **What It Is**: Digital signatures use hash functions to verify the authenticity and integrity of a message or document. The sender creates a hash of the message and encrypts it with their private key to create the digital signature.

- **Usage**: Widely used in email encryption, software distribution, and financial transactions to ensure that the data has not been tampered with.

- **Security**: By using hash functions, digital signatures provide both data integrity and non-repudiation, ensuring that the message was not altered and was sent by the claimed sender.

3. **Merkle Tree**

- **What It Is**: A Merkle Tree is a data structure used to efficiently and securely verify the integrity of data. It uses hash functions to combine and hash data in pairs, forming a tree-like structure where the final hash, known as the Merkle Root, represents the entire data set.

- **Usage**: Commonly used in blockchain technology, where it allows efficient and secure verification of large amounts of data.

- **Security**: The Merkle Tree's structure ensures that any change in the data will result in a completely different Merkle Root, making it easy to detect tampering.

Importance in Cryptography:

Cryptographic hash functions are crucial for ensuring data integrity, authenticity, and security in various applications, including digital signatures,

secure communications, and blockchain technology. They are the backbone of modern cryptographic systems.

Summary:

- **SHA-256**: A secure and widely used hash function.
- **Digital Signatures**: Ensures authenticity and integrity.
- **Merkle Tree**: Efficiently verifies large datasets.

These hash functions play a vital role in maintaining the security and integrity of data in the digital world.

Properties of Cryptographic Hash Functions - Exam Notes

Cryptographic hash functions are special functions that take an input and produce a fixed-size string of characters, which appears random. Here are the key properties of these functions:

1. **Deterministic**:
   - **What It Means**: If you put the same input into the hash function, you will always get the same output.
   - **Why It's Important**: This consistency is important for verifying data, like when you store passwords or check if files are unchanged.

2. **Pre-image Resistance**:
   - **What It Means**: Given the output (hash), it should be very hard to figure out what the original input was.
   - **Why It's Important**: This property ensures that even if someone has the hash, they cannot easily reverse it to find the original data, like a password.

3. **Avalanche Effect**:

   - **What It Means**: A small change in the input (even just one letter) will result in a completely different hash output.

   - **Why It's Important**: This makes it hard to predict what the hash will be, even if the input changes slightly.

4. **Collision Resistance**:

   - **What It Means**: It's very difficult to find two different inputs that produce the same hash.

   - **Why It's Important**: This prevents attackers from finding two different sets of data that hash to the same value, which is crucial for things like digital signatures.

5. **Fast to Compute**:

   - **What It Means**: The hash function should work quickly.

   - **Why It's Important**: Fast processing is necessary for the hash function to be practical, especially when used in real-time applications like secure websites.

6. **Fixed Output Size**:

   - **What It Means**: No matter how big or small the input is, the hash output will always be the same length.

   - **Why It's Important**: This uniformity makes it easier to store and compare hashes, and it's crucial for cryptographic systems.

Summary:

- **Deterministic**: Same input, same output.

- **Pre-image Resistance**: Hard to find the original input from the hash.

- **Avalanche Effect**: Small input change causes a big change in output.

- **Collision Resistance**: Hard to find two inputs with the same hash.

- **Fast to Compute**: The hash function works quickly.

- **Fixed Output Size**: Output is always the same length, no matter the input.

These properties make cryptographic hash functions secure and reliable for tasks like verifying data, securing passwords, and creating digital signatures.

The Role of Hashing in Blockchain - Exam Notes (5 Marks)

What is Hashing?

- Hashing is a process that takes any data (like a file or a message) and turns it into a fixed-size string of letters and numbers.

- This string is called a "hash," and it uniquely represents the original data.

How Hashing Works in Blockchain:

1. **Keeps Data Safe:**

   - Hashing ensures that the information in each block of the blockchain is secure.

   - If anyone tries to change the information in a block, the hash will change too, making it obvious that something was altered.

2. **Connects Blocks Together:**

   - Each block in a blockchain contains the hash of the previous block. This links them together in a chain.

   - If someone tries to change one block, it breaks the chain, showing that the data has been tampered with.

3. **Helps Secure the Blockchain:**

- Hashing is part of the "proof of work" system, where miners solve difficult puzzles to add new blocks to the blockchain.

- This process makes it hard for anyone to change the blockchain because it requires a lot of computing power.

4. **Protects Information:**

- Hashing makes it nearly impossible to figure out the original data just by looking at the hash.

- This keeps the information in the blockchain private and secure.

5. **Manages Large Data:**

- No matter how big the data is, hashing turns it into a fixed-size hash. This makes it easier to store and compare data in the blockchain.

Summary:

- **Safe Data:** Hashing ensures that data hasn't been changed.

- **Connected Blocks:** Blocks are linked securely in the chain.

- **Blockchain Security:** Miners use hashing to keep the blockchain secure.

- **Protects Information:** Hashing keeps data private.

- **Manages Data:** Hashing helps handle large amounts of data efficiently.

Hashing is crucial for keeping blockchain secure, reliable, and functional.

Importance of Hashing in Blockchain - Exam Notes (5 Marks)

1. **Data Integrity:**

- Hashing ensures that the data within each block has not been altered. If any data is changed, the hash value changes, making it easy to spot tampering.

2. **Connecting Blocks:**

- Hashing links blocks together in the blockchain. Each block includes the hash of the previous block, forming a secure chain. Any attempt to change a block breaks the chain, revealing tampering.

3. **Security:**

- Hashing provides a layer of security in the blockchain. Since hashes are unique to the data they represent, it's nearly impossible to predict or replicate the hash without the original data.

4. **Efficient Verification:**

- Hashes make it easy and fast to verify the data in the blockchain. Instead of checking all the data, you can simply compare hash values to ensure everything is correct.

5. **Proof of Work:**

- In many blockchains, hashing is used in the proof-of-work process, where miners solve complex puzzles to add new blocks. This helps maintain the security and trustworthiness of the blockchain.

SHA-256

Working of SHA-256 Algorithm - Exam Notes (6 Marks)

Introduction:

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that generates a 256-bit (32-byte) fixed-size hash value from input data. It is part of the SHA-2 family, developed by the National Security Agency (NSA) in the United States. SHA-256 is widely used in security protocols and applications, including SSL/TLS, digital signatures, and blockchain technology.

- **SHA-256** (Secure Hash Algorithm 256-bit) creates a unique 256-bit hash value (also called a digest) from any input data. This digest is a fixed length, even if the input data is very large. SHA-256 is essential for ensuring data integrity and is widely used in security, including in blockchain technology.

Preprocessing:

The input message is padded to ensure its length is congruent to 448 modulo 512. Padding involves appending a single '1' bit followed by enough '0' bits to make the length of the message 64 bits short of a multiple of 512. The final 64 bits of the padded message represent the length of the original message in binary.

1. **Padding the Input:**

- The input data is padded so that its length is a multiple of 512 bits. Padding adds a single '1' bit and enough '0' bits to fill up to 64 bits less than the next 512-bit boundary. Then, the original length of the input data is added as a 64-bit number.

1.Convert Data to Binary
2.Padding Input
3.Divide the Data into Blocks

2. **Dividing the Input:**

- The padded data is divided into 512-bit blocks. Each block is processed one at a time in the SHA-256 algorithm.

Hash Computation:

SHA-256 uses eight 32-bit words as initial hash values. These are constants derived from the fractional parts of the square roots of the first eight prime numbers

1. **Initializing Hash Values:**

- SHA-256 starts with eight fixed initial hash values, each 32 bits long. These values come from the fractional parts of the square roots of the first eight prime numbers.

2. **Processing Blocks:**

- Each 512-bit block is processed through 64 rounds. In each round:

  - The block is expanded into 64 smaller 32-bit words.

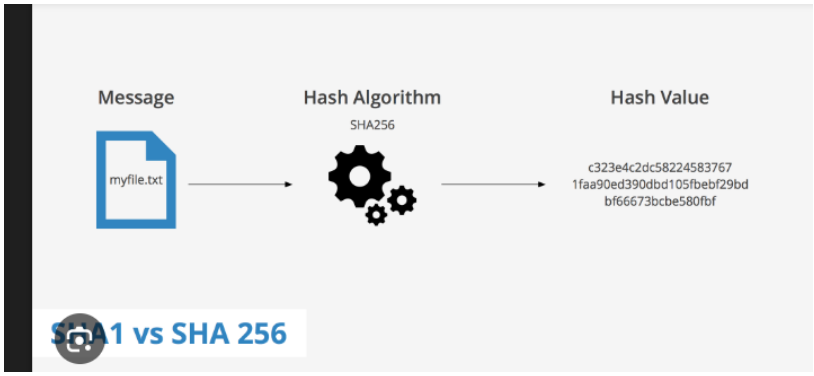
  - Mathematical and logical operations are applied using the initial hash values and the expanded words.

  - The results from each round are mixed together to update the hash values.

1.Further Division into Smaller Pieces
2.Processing Through Multiple Rounds
3. Creating New Hash Values
4. Final Hash Digest

3. **Final Hash:**

- After processing all blocks, the final hash values are combined to form the 256-bit digest. This digest is the final output of the SHA-256 algorithm.



Applications:

- **SHA-256** is used to ensure data integrity in various security applications, like digital signatures and blockchain, by generating a unique digest for any input data.

Here’s a structured note on the Digital Signature Algorithm (DSA), including the two approaches and the working of DSA, with explanations of key generation and verification:

Digital Signature Algorithm

Digital Signature Algorithm (DSA) Overview

Introduction:

The Digital Signature Algorithm (DSA) is a widely used cryptographic algorithm designed for creating and verifying digital signatures. It ensures the integrity and authenticity of messages or documents. A digital signature is a unique code attached to a message that proves the message came from the sender and hasn't been tampered with.

Two Approaches to Use DSA with Encryption

1. **Sign Then Encrypt:**

   - **Process:**

     - **Signing:** First, the sender creates a digital signature for the message using their private key. This signature is generated by hashing the message and then encrypting the hash with the sender's private key.

     - **Encrypting:** After the signature is created, the sender then encrypts the entire message (including the signature) using the recipient’s public key.

     - **Purpose:** This approach ensures that the message can only be read by the intended recipient and the recipient can verify the authenticity of the message using the sender’s public key.

2. **Encrypt Then Sign:**

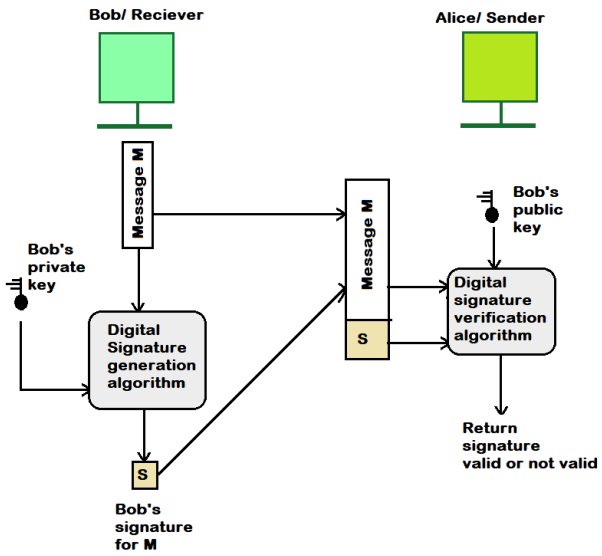
   - **Process:**

     - **Encrypting:** The sender first encrypts the message using the recipient’s public key.

     - **Signing:** After the message is encrypted, the sender creates a digital signature for the encrypted message using their private key.

     - **Purpose:** This approach provides confidentiality because only the recipient can decrypt the message. The digital signature ensures that the message came from the sender and has not been altered.

Working of DSA



1. **Key Generation:**

- **Private Key:** A secret key known only to the owner. It is used to create the digital signature.

- **Public Key:** A key that can be shared with others. It is used to verify the digital signature.

- **Steps:**

1. **Generate a large prime number (p) and a corresponding base (g).**

2. **Choose a private key (x) and compute the public key (y) using (y = g^x mod p).**

3. **The public key ((p, g, y)) and the private key (x) are used in the signature process.**

2. **Signature Generation:**

- **Hash the Message:** Compute a hash (H(m)) of the message (m) using a hash function.

- **Generate a Random Value** (k):** Choose a random number (k) for each signature.

- **Compute (r) and (s):** Use the private key and (k) to compute (r) and (s), which together form the signature.

3. **Signature Verification:**

   - **Hash the Received Message:** Compute the hash (H(m)) of the received message.

   - **Check the Signature:** Use the public key to verify that the signature is valid for the given message hash. This involves several mathematical checks using the values (r), (s), and the public key.

Conclusion

The Digital Signature Algorithm provides a way to ensure that messages are authentic and have not been tampered with. By using either the "Sign Then Encrypt" or "Encrypt Then Sign" approach, users can maintain the integrity and confidentiality of their communications.

Advantages and Disadvantages of Digital Signature Algorithm (DSA)

Advantages

1. **Security:**

   - **Strong Authentication:** DSA provides strong authentication, ensuring that a signature is genuine and originates from the legitimate sender.

   - **Tamper-Proof:** The digital signature ensures that the message has not been altered during transit.

2. **Integrity:**

- **Message Integrity:** By using a hash function, DSA ensures that any changes to the message will result in a different hash, thereby revealing tampering.

3. **Efficiency:**

- **Efficient Signing Process:** DSA is designed to be efficient in generating signatures, which can be useful in environments where performance is crucial.

4. **Non-Repudiation:**

- **Proof of Origin:** Since the digital signature is unique to the sender's private key, the sender cannot deny sending the message once it has been signed.

5. **Widely Adopted:**

- **Standardization:** DSA is a well-established standard used in various security protocols and systems, including government and industry applications.

Disadvantages

1. **Performance:**

- **Signing vs. Verification Speed:** While signing is efficient, verification can be slower compared to some other algorithms, which might affect performance in high-load scenarios.

2. **Key Management:**

- **Private Key Security:** The security of the system depends heavily on the protection of the private key. If the private key is compromised, the entire system's integrity is at risk.

3. **Lack of Forward Secrecy:**

- **No Forward Secrecy:** Unlike some modern encryption schemes, DSA does not inherently provide forward secrecy, which means if private keys are compromised, past communications can potentially be decrypted.

4. **Vulnerability to Certain Attacks:**

- **Weak Random Number Generators:** DSA's security relies on the randomness of the value \(k \). Weak or predictable random number generators can lead to vulnerabilities.

5. **Key Size Requirements:**

- **Key Size vs. Security:** To maintain security, DSA requires relatively large key sizes, which can impact performance and resource usage.

Comparison Between DSA and RSA

1. Purpose and Function

- **DSA (Digital Signature Algorithm):**

- **Primary Use:** Designed specifically for digital signatures.

- **Function:** Ensures message authenticity and integrity by creating a unique signature with the sender's private key, verifiable by anyone with the sender's public key.

- **RSA (Rivest-Shamir-Adleman):**

- **Primary Use:** Used for both encryption and digital signatures.

- **Function:** Can encrypt data and create digital signatures using modular arithmetic.

2. Key Generation

- **DSA:**

  - **Key Pair:** Includes a private key for signing and a public key for verification.

  - **Basis:** Relies on the difficulty of the discrete logarithm problem.

- **RSA:**

  - **Key Pair:** Includes a private key for decryption/signing and a public key for encryption/verification.

  - **Basis:** Relies on the difficulty of factoring large numbers.

3. Algorithmic Approach

- **DSA:**

  - **Hash Function:** Uses hash functions (like SHA-2) to process the message before signing.

  - **Signature Generation:** Creates a signature from the hashed message and a random number.

- **RSA:**

  - **Encryption/Signing:** Operates directly on the message or its hash using modular arithmetic.

4. Security and Performance

- **DSA:**

  - **Advantages:** Generally faster for signing operations.

  - **Disadvantages:** Verification can be slower; requires secure random number generation.

- **RSA:

  - **Advantages:** Versatile (can be used for both encryption and signing).

  - **Disadvantages:** Slower performance; requires larger keys for comparable security.

b) Explain DSA key generation and verification. [5]

b) Describe Digital Signature & Verification steps in Digital Signature Algorithm. [4]

Here's a breakdown of the steps involved in Digital Signature Generation and Verification using the Digital Signature Algorithm (DSA):

Digital Signature in DSA

A **Digital Signature** is like a digital stamp that confirms the sender's identity and ensures that the message hasn't been altered. It is created using the sender's private key and can be verified by anyone with the sender's public key.

1. Digital Signature Generation

1. **Message Hashing:**
   - **Step:** First, the sender applies a cryptographic hash function (like SHA-256) to the message. This produces a hash value (a fixed-size string) that uniquely represents the message.
   - **Purpose:** The hash function converts the message into a compact representation, making it easier to handle.
2. **Signature Creation:**
   - **Step:** The sender uses their private key and the hash value to create the digital signature. This involves two main operations:
     - **Generating rrr:** Compute rrr using a random number and the private key.
     - **Generating sss:** Compute sss by combining the hash value with the private key and rrr.
   - **Purpose:** The digital signature (rrr, sss) is a pair of values that prove the authenticity and integrity of the message.

2. Digital Signature Verification

1. **Message Hashing:**

- **Step:** The recipient applies the same cryptographic hash function to the received message, producing a hash value.

2. **Signature Validation:**
   - **Step:** The recipient uses the sender's public key, along with the received signature (rrr, sss), to verify the digital signature.
     - **Compute Hash:** Calculate the expected hash value from the signature.
     - **Compare:** Check if this computed hash value matches the hash value from the received message.
   - **Purpose:** Verification ensures that the signature was created using the sender's private key and that the message has not been tampered with.

In Summary:

- **Signature Generation:** Hash the message, then create a signature using the private key.
- **Signature Verification:** Hash the received message, then use the sender's public key to check if the signature matches the hash.

Q1)	a)	List and explain advantages of ECC.	**[5]**
	b)	Differentiate asymmetric and symmetric key cryptography.	**[5]**
	c)	Discuss the properties of hash function.	**[5]**
		OR	
Q2)	a)	List and discuss benefits of Merkel tree.	**[5]**
	b)	Explain DSA key generation and verification.	**[5]**
	c)	Discuss role of hashing in Blockchain.	**[5]**

Q1)	a)	Illustrate Elliptic Curve Gyptography.	**[6]**
	b)	Justify the importance of Hashing in Block Chain.	**[4]**
	c)	What is Merkle tree? Explain the structure of merkle tree.	**[5]**
		OR	
Q2)	a)	Explain working of SHA 256 Algorithm.	**[6]**
	b)	Describe Digital Signature & Verification steps in Digital Signature Algorithm.	**[4]**
	c)	Describe Symmetric Key Encryption with neat diagram.	**[5]**

1. Convert Data to Binary
Everything in computers is stored as binary code, which is a series of 0s and 1s. For example, the letter "a" is represented as "01000001" in binary.
2. Divide the Data into Blocks
The binary data is broken down into chunks of 512 bits. If the data is less than 512 bits, it's padded with extra bits to make it exactly 512 bits. If it's more, it's split into multiple 512-bit blocks.
3. Further Division into Smaller Pieces
Each 512-bit block is then divided into smaller chunks of 32 bits.
4. Processing Through Multiple Rounds
The main part of SHA-256 involves processing each 512-bit block through 64 rounds of calculations. In each round, the data is mixed up and transformed in specific ways. These rounds ensure that the final hash is very different, even if the original data changes slightly.
5. Creating New Hash Values
As the data goes through each round, new intermediate hash values are created. These values are combined and modified in every round.
6. Final Hash Digest
After all 64 rounds, the final output is a 256-bit hash value. This is the "fingerprint" of the original data. No matter how large or small your input was, the output is always a 256-bit hash.
This final hash is unique to the input data. Even a tiny change in the input will result in a completely different hash. That's why SHA-256 is widely used in security, as it's nearly impossible to reverse-engineer the original data from the hash or to find two different pieces of data that produce the same hash.