**1. Program for Default, Printed, and Plaintext Passwords**

**Algorithm:**

1. Start
2. Set a default password.
3. Print default password (hidden).
4. Accept a user-defined password (hidden input).
5. Print both the default and entered passwords in plain text.
6. End

**Flowchart:**

I can provide a flowchart image—would you like me to generate that?

**Python Code:**

python

CopyEdit

import getpass

# Step 1: Default Password

default\_password = "Admin@123"

# Step 2: Print default password (masked)

print("Default password is set.")

# Step 3: User enters new password (masked input)

user\_password = getpass.getpass("Enter a new password: ")

# Step 4: Show both passwords in plain text

print("\n--- Passwords in Plain Text ---")

print("Default Password:", default\_password)

print("User Password :", user\_password)

**Sample Output:**

pgsql

CopyEdit

Default password is set.

Enter a new password: \*\*\*\*\*\*\*\*

--- Passwords in Plain Text ---

Default Password: Admin@123

User Password : Secret@456

**2. RSA Algorithm with Example**

**Steps:**

1. Choose two primes: p = 3, q = 11
2. Compute n = p × q = 33
3. Compute φ(n) = (p−1)(q−1) = 2×10 = 20
4. Choose e = 7 such that 1 < e < φ(n) and gcd(e, φ(n)) = 1
5. Compute d such that (d × e) mod φ(n) = 1 → d = 3
6. Public key: (e, n) = (7, 33)
7. Private key: (d, n) = (3, 33)

**Encryption:**  
Message M = 4 → Ciphertext C = M^e mod n = 4^7 mod 33 = 16384 mod 33 = 16

**Decryption:**  
Ciphertext C = 16 → Plaintext M = C^d mod n = 16^3 mod 33 = 4096 mod 33 = 4

**3. Any Two Classical Ciphers**

**i. Playfair Cipher:**

* Uses a 5x5 matrix with a keyword.
* Encrypts digraphs (pair of letters).
* Example:
  + Key: MONARCHY, Message: HELLO
  + Encrypted: HE → DM, LL → MP, OX → NA

**ii. Vigenère Cipher:**

* Uses a keyword to shift letters (like multiple Caesar ciphers).
* Example:
  + Plaintext: ATTACK
  + Key: LEMON
  + Encrypted: LXFOPV

**4. Comparison Table**

| **Technique** | **Confidentiality** | **Integrity** | **Authentication** | **Non-repudiation** |
| --- | --- | --- | --- | --- |
| Classical Cipher | Yes | No | No | No |
| Symmetric Encryption | Yes | No | Yes (if key shared securely) | No |
| Asymmetric Encryption | Yes | No | Yes | Yes |
| Hashing | No | Yes | No | No |
| MAC | No | Yes | Yes | No |
| Digital Signature | No | Yes | Yes | Yes |

**5. DES Algorithm (Data Encryption Standard)**

* Block cipher, 64-bit input, 56-bit key.
* 16 rounds of Feistel structure:
  1. Initial Permutation
  2. 16 rounds of substitution/permutation using round keys
  3. Final Permutation
* Uses S-boxes, expansion, XOR, and permutation.

Example:

* Plaintext: 0x0123456789ABCDEF
* Key: 0x133457799BBCDFF1
* Ciphertext: 0x85E813540F0AB405

**6. Classical vs Modern Cryptography**

| **Feature** | **Classical Cryptography** | **Modern Cryptography** |
| --- | --- | --- |
| Type | Character-based | Bit-based |
| Security | Relatively low | Strong with complex math |
| Key | Mostly symmetric | Symmetric & Asymmetric |
| Example | Caesar, Vigenère | AES, RSA, ECC |

**7. DoS vs DDoS Attack and Mitigation**

* **DoS (Denial of Service):** Overloads a system with requests to disrupt service.
* **DDoS (Distributed DoS):** Uses multiple compromised machines to flood the target.

**Mitigation:**

* Use firewalls and rate-limiting.
* Implement Intrusion Detection Systems (IDS).
* Use Content Delivery Networks (CDNs).
* Cloud-based DDoS protection services like Cloudflare or AWS Shield.

Here’s how you can tackle the questions:

**Q8 & Q9: Transposition Cipher Program**

**Algorithm**

1. Choose a key (e.g., *MEGABUCK*, *PICTENTG*, *NBAISOKR*).
2. Remove spaces from the message and pad if needed.
3. Create a grid with columns = length of the key.
4. Fill in the message row-wise.
5. Sort the key alphabetically and number columns accordingly.
6. For encryption, read columns in the key’s alphabetical order.
7. For decryption, reverse the process using column order.

**Python Program**

def encrypt\_transposition(message, key):

key\_order = sorted(list(enumerate(key)), key=lambda x: x[1])

col\_order = [i[0] for i in key\_order]

num\_cols = len(key)

num\_rows = -(-len(message) // num\_cols) # Ceiling division

padded\_len = num\_rows \* num\_cols

message += 'X' \* (padded\_len - len(message)) # Pad with 'X'

grid = [list(message[i:i+num\_cols]) for i in range(0, padded\_len, num\_cols)]

cipher = ''.join([''.join([row[i] for row in grid]) for i in col\_order])

return cipher

def decrypt\_transposition(cipher, key):

key\_order = sorted(list(enumerate(key)), key=lambda x: x[1])

col\_order = [i[0] for i in key\_order]

num\_cols = len(key)

num\_rows = len(cipher) // num\_cols

cols = [''] \* num\_cols

k = 0

for index in col\_order:

cols[index] = cipher[k:k+num\_rows]

k += num\_rows

plaintext = ''

for i in range(num\_rows):

for j in range(num\_cols):

plaintext += cols[j][i]

return plaintext.rstrip('X') # Remove padding

# Example

message = "pleasetransferonemilliondollarstomyswissbankaccountsixtwo twofour".replace(" ", "")

key = "MEGABUCK"

cipher = encrypt\_transposition(message, key)

print("Encrypted:", cipher)

print("Decrypted:", decrypt\_transposition(cipher, key))

**Flowchart**

Would you like a **generated flowchart image** for encryption and decryption logic?

**Q10: C, I, A, Authentication, Non-Repudiation**

| **Term** | **Explanation** |
| --- | --- |
| **Confidentiality** | Ensuring data is not accessed by unauthorized parties. |
| **Integrity** | Ensuring data is accurate and untampered. |
| **Availability** | Ensuring systems/data are accessible when needed. |
| **Authentication** | Verifying identity of users/systems. |
| **Non-Repudiation** | Ensures a sender cannot deny sending a message. |

**Q11: Types of Attacks**

| **Attack Type** | **Description** | **Relates to** |
| --- | --- | --- |
| **Interruption** | Service/data becomes unavailable. | A (Availability) |
| **Interception** | Unauthorized access to data. | C (Confidentiality), Authentication |
| **Modification** | Altering data in transit. | I (Integrity), Authentication |
| **Fabrication** | Fake data/information is created. | I, Authentication, Non-Repudiation |

**Q15 & Q16: Caesar Cipher Program**

**Algorithm**

1. Convert message to lowercase and remove spaces (or handle them as needed).
2. Choose key (shift value).
3. For each character:
   * If it's a letter, shift it by the key value in the alphabet (wrap around using modulo).
4. For decryption, shift in the reverse direction.

**Python Program**

def encrypt\_caesar(message, key):

result = ""

for char in message.lower():

if char.isalpha():

shifted = (ord(char) - ord('a') + key) % 26

result += chr(shifted + ord('a'))

else:

result += char

return result

def decrypt\_caesar(cipher, key):

return encrypt\_caesar(cipher, -key)

# Test with message and different keys

message = "please transfer one million dollars to my swiss bank account six two two four"

keys = [3, 5, 7]

for k in keys:

encrypted = encrypt\_caesar(message, k)

decrypted = decrypt\_caesar(encrypted, k)

print(f"\nKey = {k}")

print("Encrypted:", encrypted)

print("Decrypted:", decrypted)

**Flowchart**

Would you like me to generate a flowchart for Caesar Cipher encryption/decryption?

**Q20: Two Classical Ciphers**

**i. Vigenère Cipher**

* Uses keyword for encryption.
* Each letter is shifted by the position of the corresponding keyword letter.
* Example:
  + Message: HELLO, Key: KEY
  + H + K → R, E + E → I, L + Y → J, ...

**ii. One-Time Pad**

* Uses a random key of the same length as the message.
* Offers perfect secrecy.
* Example:
  + Message: CAT, Key: XQZ
  + Encrypt by XORing ASCII or letter positions.

**iii. Monoalphabetic Cipher**

* Each plaintext letter maps to a fixed ciphertext letter (simple substitution).
* Key is a 26-letter shuffled alphabet.

Here's a detailed breakdown and answer guide for your cryptography and wireless security questions:

**Q22. Demonstrate installation and configuration of mobile security app.**

**Example App**: **Avast Mobile Security** (for Android)

**Steps:**

1. **Installation:**
   * Open Google Play Store.
   * Search for “Avast Mobile Security.”
   * Click *Install* and wait for the app to download.
2. **Configuration:**
   * Open the app after installation.
   * Accept permissions (like access to storage, contacts, etc.).
   * Enable real-time scanning and anti-theft features.
   * Configure app locking, call blocking, and VPN settings.

**Features:**

* Real-time malware scanning.
* Wi-Fi security checks.
* App privacy advisor.
* VPN and web shield.
* Anti-theft and device location.

**Snapshots to Record:**

* App download page on Play Store.
* First launch screen with permissions.
* Home dashboard with security status.
* Scan result screen.
* VPN connection screen.
* Anti-theft setup screen.

**Q23. What is WEP and WAP security techniques?**

* **WEP (Wired Equivalent Privacy):**
  + An early security algorithm for IEEE 802.11 wireless networks.
  + Uses RC4 stream cipher for confidentiality.
  + Employs 40-bit or 104-bit keys.
  + Weak due to poor IV management and key reuse.
* **WAP (Wireless Application Protocol):**
  + Not a direct security protocol, but includes **WTLS (Wireless Transport Layer Security)** for data encryption.
  + Used in early mobile internet access (like WAP browsers).
  + Provides authentication, data integrity, and privacy over wireless networks.

**Q24. Wireless components used for Wi-Fi and Bluetooth communications:**

* **Wi-Fi Components:**
  + Wireless Access Point (WAP)
  + Wireless Routers
  + Wireless Network Interface Cards (NIC)
  + Antennas (Omni-directional or Directional)
  + Repeaters and Range Extenders
* **Bluetooth Components:**
  + Bluetooth Transceivers (in phones, headsets, etc.)
  + Bluetooth Dongles for PCs
  + Embedded modules in IoT devices
  + Antennas for short-range communication

**Q25. ISM Band Frequencies, BT Standards & Wi-Fi Standards**

* **ISM Band Frequencies:**
  + 2.4 GHz (common for Wi-Fi and Bluetooth)
  + 5.8 GHz
  + 902–928 MHz (less common)
* **Bluetooth Standards:**
  + BT 1.2: Improved interference handling
  + BT 2.0 + EDR: Faster data rate (3 Mbps)
  + BT 4.0: Introduced BLE (Bluetooth Low Energy)
  + BT 5.0+: Increased range, speed, and broadcasting
* **Wi-Fi Standards:**
  + 802.11a: 5 GHz, up to 54 Mbps
  + 802.11b: 2.4 GHz, up to 11 Mbps
  + 802.11g: 2.4 GHz, up to 54 Mbps
  + 802.11n: Dual-band, up to 600 Mbps
  + 802.11ac: 5 GHz, Gigabit speeds
  + 802.11ax (Wi-Fi 6): Higher efficiency and capacity

**Q26. Compare Symmetric and Asymmetric Key Cryptography (Min 8 Points)**

| **Feature** | **Symmetric Key** | **Asymmetric Key** |
| --- | --- | --- |
| Keys Used | Single key (shared) | Public and private key pair |
| Speed | Faster | Slower |
| Complexity | Less complex | More complex |
| Key Distribution | Challenging | Easier (public key can be shared) |
| Encryption & Decryption Speed | High | Lower |
| Common Algorithms | AES, DES, RC4 | RSA, ECC |
| Suitable For | Bulk data encryption | Key exchange, digital signatures |
| Security Level | Lower if key is exposed | Higher security |

**Q27. DES Algorithm – Diagram and Explanation**

**Diagram**: [Would you like me to provide a diagram image for DES?]

**Explanation:**

* DES (Data Encryption Standard) is a symmetric key block cipher.
* Operates on 64-bit blocks with a 56-bit key.
* **Main steps:**
  1. Initial Permutation (IP)
  2. 16 Rounds of Feistel structure:
     + Expansion (E)
     + Key mixing (XOR with round key)
     + Substitution (S-boxes)
     + Permutation (P)
  3. Final Permutation (Inverse IP)

**Q28. Transport and Tunnel Mode in IPSec**

* **Transport Mode:**
  + Encrypts only the **payload** of the IP packet.
  + Used for end-to-end communication (e.g., host-to-host).
  + IP header remains unchanged.
* **Tunnel Mode:**
  + Encrypts the **entire IP packet** (payload + header).
  + Adds a new IP header for routing.
  + Used in **VPNs** (e.g., gateway-to-gateway or host-to-gateway).

| **Mode** | **What is Encrypted** | **Common Use Case** |
| --- | --- | --- |
| Transport | Only data/payload | Host-to-host communication |
| Tunnel | Entire packet | Site-to-site VPN |

**Q29. Demonstrate installation and configuration of Steganography technique**

**Tool Used**: **OpenPuff or SilentEye (Windows)** | **Steghide (Linux)**

**Steps using SilentEye (Windows GUI Tool):**

1. **Download & Install:**
   * Download from <https://sourceforge.net/projects/silenteye/>
   * Install and open the software.
2. **Configuration:**
   * Select image/audio file as the cover medium.
   * Enter the message or file to be hidden.
   * Choose encryption method (AES, DES, etc.).
   * Set password (for optional encryption).
   * Save the stego-file.
3. **Decoding:**
   * Load the stego-file.
   * Enter password (if encryption used).
   * Extract hidden message or file.

**Features:**

* Supports JPEG and BMP (for images).
* AES and DES encryption.
* Simple GUI for encoding/decoding.
* Password protection.

**Snapshots to Record:**

* Home screen of SilentEye.
* Image selection screen.
* Message embedding screen.
* Stego-image generation.
* Extraction screen showing message retrieval.

**Q30. Draw and explain the block diagram of Steganography for Image as Data**

**[Diagram Description – Optional: I can provide an image if needed]**

**Explanation:**

1. **Input Data** (Message/File to hide)
2. **Cover Image** (e.g., JPEG, PNG)
3. **Steganographic Encoder**:
   * Embeds the message in the image using techniques like LSB (Least Significant Bit)
   * Optional: encryption before embedding
4. **Stego Image** (output file with hidden data)
5. **Steganographic Decoder**:
   * Extracts hidden data from stego image
   * Decrypts (if encrypted)

**Q31. Compare Steganography vs Cryptography**

| **Feature** | **Steganography** | **Cryptography** |
| --- | --- | --- |
| Purpose | Hides the *existence* of a message | Hides the *content* of a message |
| Visibility | Invisible to third parties | Visible encrypted text (ciphertext) |
| Detection | Harder to detect | Easy to detect (though unreadable) |
| Use in Network Sec. | Used for covert communication | Used for secure communication |
| Example | Hiding text in images or audio | Encrypting emails or files |
| Security dependency | Depends on secrecy of method | Depends on strength of algorithm/key |
| Combination | Can be combined with crypto for security | Often combined with steganography |
| Tamper Detection | Less robust | Stronger with hashing, signatures |

**Q32. Draw and explain how DES algorithm works in detail**

**[Diagram can be provided as an image if requested]**

**Explanation:**

* **Input**: 64-bit plaintext and 56-bit key
* **Steps**:
  1. **Initial Permutation (IP)**
  2. Split into L0 and R0 (32 bits each)
  3. 16 Rounds of:
     + Expansion (E) from 32 to 48 bits
     + XOR with round key
     + Substitution via 8 S-boxes
     + Permutation
     + XOR with left half
     + Swap halves
  4. Final Permutation (Inverse IP)
* **Output**: 64-bit ciphertext

**Q33. Explain RSA Algorithm with Example**

**RSA (Rivest-Shamir-Adleman)** is an asymmetric key cryptographic algorithm.

**Steps:**

1. **Key Generation**:
   * Choose two primes: p = 7, q = 11
   * Compute n = p × q = 77
   * Compute φ(n) = (p−1)(q−1) = 60
   * Choose e = 13 (1 < e < φ(n), and gcd(e, φ(n)) = 1)
   * Compute d such that (d × e) mod φ(n) = 1 → d = 37

**Public Key**: (e=13, n=77)  
**Private Key**: (d=37, n=77)

1. **Encryption**:
   * Message M = 9
   * Ciphertext C = M^e mod n = 9^13 mod 77 = 57
2. **Decryption**:
   * M = C^d mod n = 57^37 mod 77 = 9

**Q35. Draw and explain the following block diagrams**

**a. Digital Signature System**

**Diagram Flow:**

Sender

|

Message → Hash Function → Hash → Encrypt with Private Key → Digital Signature

|

Send: Message + Signature

Receiver

|

Verify Signature using Sender’s Public Key

Compare with Hash of Received Message

**Explanation:**

* Sender generates a hash of the message.
* Hash is encrypted with the sender’s private key (signature).
* Receiver uses sender’s public key to decrypt signature and compares it with hash of the received message to verify authenticity.

**b. End-to-End Email Communication System**

**Diagram Flow**:

Sender:

Message → Hash → Signature (Private Key)

↓

Encrypt with Receiver's Public Key → Digital Envelope

↓

Send Email (Message + Signature + Envelope)

Receiver:

Decrypt Envelope with Private Key → Get Message

Verify Hash with Sender’s Public Key

**Explanation:**

* Sender signs message hash (digital signature).
* Message + signature is encrypted using receiver’s public key (digital envelope).
* Receiver decrypts using private key and verifies signature using sender’s public key.

**Q36. Install and Configure Firewall for Host Security**

**Tool Example**: **Comodo Firewall (Windows)** or **UFW (Linux)**

**Steps (for Windows using Comodo Firewall):**

1. **Installation**:
   * Download from official site.
   * Install and reboot the system if required.
2. **Configuration**:
   * Launch Comodo.
   * Enable Firewall protection.
   * Set up custom rules (block port, IP, or app).
   * Enable intrusion detection and alerts.

**Features:**

* Packet filtering
* Application control
* Port blocking
* Stealth ports (to hide from port scanners)
* Logs and alerts
* Sandbox for unknown applications

**Snapshots to Record:**

* Installation screen
* Firewall dashboard
* Custom rule creation (e.g., blocking a port)
* Alerts/logs from suspicious activity

**Q37. Draw and Explain**

**a) Packet Filtering Firewall**

* **Diagram**:

[Internet] → [Packet Filter Firewall] → [Internal Network]

* **Explanation**:
  + Operates at **Network Layer**.
  + Filters based on IP addresses, ports, and protocols.
  + Stateless or basic stateful inspection.

**b) Application Layer Firewall**

* **Diagram**:

[Internet] → [Application Firewall] → [App Server]

* **Explanation**:
  + Works at **OSI Layer 7**.
  + Filters traffic based on HTTP, FTP, DNS, etc.
  + Can detect specific app-layer attacks like SQL injection.

**c) Circuit Level Gateway Firewall**

* **Diagram**:

[Client] ↔ [Circuit-level Firewall] ↔ [Internet]

* **Explanation**:
  + Works at **Session Layer**.
  + Monitors TCP handshakes.
  + Ensures connection legitimacy without inspecting content.

**Q38. Compare Firewall vs Antivirus**

| **Feature** | **Firewall** | **Antivirus** |
| --- | --- | --- |
| Purpose | Blocks unauthorized network access | Detects/removes malicious software |
| Layer Operated | Network Layer | File/System Layer |
| Functionality | Filters traffic | Scans files and applications |
| Protection Type | Prevents external threats | Cleans internal threats |
| Example | Comodo, Windows Defender Firewall | Avast, McAfee, Norton |
| Signature Based? | No (rule/policy based) | Yes (scans with virus signatures) |
| Real-Time Scan | Of data packets | Of files & memory |
| Complementary? | Yes, works with antivirus | Yes, needs firewall for network layer |

**Q39. Compare IDS and IPS in Detail**

| **Feature** | **IDS (Intrusion Detection System)** | **IPS (Intrusion Prevention System)** |
| --- | --- | --- |
| Function | Detects and alerts | Detects and blocks/thwarts threats |
| Position | Passive (after traffic) | Inline (active, between source & target) |
| Action | Alert/log | Block/terminate session |
| False Positives | Possible | Can interrupt legitimate traffic |
| Modification to Traffic | No | Yes |
| Example Tools | Snort (IDS mode), Suricata | Snort (inline mode), Cisco IPS |
| Use Case | Monitoring, forensics | Real-time attack mitigation |

Here's a complete and well-structured explanation for your **web security and email cryptography** related questions (Q43 to Q49), including practical browser configurations, security techniques, and block diagram explanations:

**Q43. Demonstrate Security Configuration in Google Chrome**

**A) Cookies Settings:**

1. **Steps**:
   * Open Chrome → Settings → Privacy and Security → **Cookies and other site data**
   * Choose:
     + Allow all cookies
     + Block third-party cookies
     + Block all cookies
2. **Feature**:
   * Protects against cross-site tracking and session hijacking.

**B) Website Blocking:**

1. **Via Extension**: Install **"BlockSite"** Chrome extension.
   * Go to Chrome Web Store → Add "BlockSite"
   * Add websites to block list
2. **Use Case**: Prevent access to malicious or distracting sites.

**C) Phrase/Word Blocking:**

* Install **"Block Site"** or **"TinyFilter"** extension.
* Configure keywords in the filter section (e.g., "gambling", "torrent")
* It blocks pages containing those keywords.

**Snapshots to Record:**

* Chrome Settings → Cookies
* BlockSite Extension → Block list
* TinyFilter or BlockSite keyword settings

**Q44. Types of Cookies**

1. **Session Cookies**:
   * Temporary; deleted after browser closes.
2. **Persistent Cookies**:
   * Stored on disk; survive browser restarts.
3. **Secure Cookies**:
   * Transmitted only over HTTPS.
4. **HttpOnly Cookies**:
   * Inaccessible via JavaScript; defends against XSS.
5. **Third-party Cookies**:
   * Set by domains other than the site being visited.
6. **Zombie Cookies**:
   * Recreated even after deletion (via Flash or backups).

**Q45. Advantages and Drawbacks of Cookies**

| **Aspect** | **Advantage** | **Drawback** |
| --- | --- | --- |
| Usability | Saves login and preferences | Tracks user behavior |
| Performance | Speeds up browsing | Increases browser storage usage |
| Personalization | Tailors content (ads, layout) | Privacy concern (especially third-party) |
| Persistence | Supports “Remember Me” features | Can be misused for session hijacking |

**Q46. Explain Session Hijacking via Cookies**

**Session Hijacking**: An attack where an attacker steals a user’s session ID stored in cookies to impersonate them.

**Methods:**

* **Packet sniffing** (in unencrypted HTTP traffic)
* **Cross-site scripting (XSS)**: Steals cookies via JavaScript
* **Man-in-the-middle (MITM)**: Intercepts cookies in transit

**Prevention:**

* Use HTTPS (TLS)
* Set **HttpOnly** and **Secure** flags
* Implement **session timeouts** and **re-authentication**

**Q47. Draw and Explain**

**a) Digital Signature System**

Sender:

Message → Hash → Encrypt with Private Key → Signature

↓

Send: Message + Signature

Receiver:

Message + Signature → Hash Message

→ Decrypt Signature (Sender’s Public Key)

→ Compare Hashes → Validate

**b) End-to-End Email Communication with Hash, Digital Signature, and Envelope**

Sender:

Message → Hash → Sign (Private Key) → Signature

Message + Signature → Encrypt with Receiver's Public Key → Digital Envelope

Send: Envelope

Receiver:

Decrypt Envelope (Private Key) → Get Message + Signature

Hash Message → Verify with Decrypted Signature (Sender’s Public Key)

Would you like the diagrams as images?

**Q49. TLS and S/MIME + Compare PGP vs S/MIME**

**TLS (Transport Layer Security):**

* **Secures** emails in **transit**.
* Encrypts the connection between email servers or user-client.
* Common in HTTPS, SMTP over TLS.

**S/MIME (Secure/Multipurpose Internet Mail Extensions):**

* **End-to-end encryption** of email content.
* Uses **X.509 certificates** for public key infrastructure.
* Provides **encryption, integrity, and authentication**.

**PGP vs S/MIME Comparison**

| **Feature** | **PGP** | **S/MIME** |
| --- | --- | --- |
| Key Management | Decentralized (Web of Trust) | Centralized (Certificate Authority) |
| Format | ASCII-armored | Binary (PKCS#7 format) |
| Certificate Type | Custom/Public keyrings | X.509 certificates |
| User-Friendly | Less user-friendly | Built into many email clients |
| Flexibility | More control by user | More enterprise-oriented |
| Used In | Open-source & personal use | Enterprises, corporations |

**Q50. Implement Hash Function Technique for Secured Network**

**Tool Used: HashCalc / Online tools like OnlineMD5, CyberChef, MD5File.com**

**Steps:**

1. Open a hash tool or visit: <https://emn178.github.io/online-tools/sha256.html>
2. Input a text or file (e.g., a password, document)
3. Select desired algorithm: **MD5**, **SHA1**, **SHA256**, etc.
4. Generate hash → Validate by rehashing same input

**Features:**

* One-way hashing
* Fixed-length output
* Fast computation
* Supports multiple hash algorithms

**Snapshots to Record:**

* Input data
* Selected algorithm
* Resulting hash
* Re-validation of hash for data integrity

**Q51. Applications of Hash Functions**

**a. Password Protection:**

* Store **hashed version** of password in database.
* During login, entered password is hashed and compared with stored hash.
* Example: bcrypt, SHA-256 + salt

**b. Data Integrity Check:**

* File or message → Hash value generated
* Receiver recalculates hash after receiving
* Mismatch → Tampering detected
* Common in software downloads, emails

**Q52. Comparison Table – Security Techniques**

| **Technique** | **Confidentiality** | **Integrity** | **Authentication** | **Non-Repudiation** |
| --- | --- | --- | --- | --- |
| Classical Encryption | ✅ | ❌ | ❌ | ❌ |
| Symmetric Encryption | ✅ | ✅ | ✅ (with key) | ❌ |
| Asymmetric Encryption | ✅ | ✅ | ✅ | ✅ |
| Hashing | ❌ | ✅ | ❌ | ❌ |
| MAC (Message Auth Code) | ❌ | ✅ | ✅ | ❌ |
| Digital Signature | ❌ | ✅ | ✅ | ✅ |

**Q53. Hash Function – Explanation**

**What is Hash Function?**

A **hash function** converts input data into a fixed-length string of characters, known as a **hash value**.

**How it Works:**

* Input → Processed through a hashing algorithm → Fixed-length output
* **Deterministic**: same input always gives same output
* **Irreversible**: cannot reverse hash to get original input

**Applications of SHA-2:**

* Digital Signatures
* SSL/TLS Certificates
* Bitcoin and Blockchain
* Password Storage
* Software Integrity Verification

SHA-2 Variants:

* SHA-224, SHA-256, SHA-384, SHA-512

**Q54. IPSec: Transport vs Tunnel Mode**

| **Feature** | **Transport Mode** | **Tunnel Mode** |
| --- | --- | --- |
| Header Encrypted | Only **payload** encrypted | **Entire packet** encrypted |
| Use Case | End-to-end between hosts | Between gateways (VPNs) |
| IP Header Visible | Yes | No |
| Example | Host A ↔ Host B | Client ↔ Gateway ↔ Internet |

**Diagram (Transport)**:

[Original IP Header][Encrypted Payload]

**Diagram (Tunnel)**:

[New IP Header][Encrypted Original IP + Payload]

**Q57. Simulate Diffie-Hellman Key Exchange (Using VLab)**

**Steps (Use Vlabs Link):**

Visit: <https://cse29-iiith.vlabs.ac.in/exp3/index.html>

1. Choose a **prime number p** and a **primitive root g**.
2. Alice selects private key a, Bob selects private key b.
3. Alice computes A = g^a mod p, Bob computes B = g^b mod p.
4. They exchange A and B.
5. Both compute shared key:  
   Alice: K = B^a mod p, Bob: K = A^b mod p.

**Features:**

* Enables **secure key exchange** over an insecure channel.
* Based on **discrete logarithm problem**.
* Used in VPNs, HTTPS, and SSH.

**Snapshots to Record:**

* Initial parameters (p, g, a, b)
* Computed values (A, B)
* Shared secret key K (same for both)

**Q58. Draw and Explain DES Algorithm in Detail**

**DES = Data Encryption Standard**

**Block Diagram Overview:**

Plaintext → Initial Permutation → 16 Rounds (Feistel Structure) → Final Permutation → Ciphertext

**Process:**

1. 64-bit plaintext input
2. Initial permutation (IP)
3. 16 rounds:
   * Each round:
     + Divide into Left & Right halves
     + Right → Expansion (32→48 bits)
     + XOR with round key (from key schedule)
     + S-box substitution → 32 bits
     + Permutation → XOR with Left half
     + Swap
4. Combine halves → Final permutation → Ciphertext

**Q60. What is Hash Function? How it Works? Applications of SHA2**

**Definition:**

A hash function takes input of any length and produces a **fixed-length hash value**.

**How it Works:**

* Input → Processed using a mathematical function → Output (e.g., 256-bit digest for SHA-256)

**SHA2 Applications:**

* Password hashing (bcrypt + SHA256)
* Blockchain and cryptocurrency
* Digital signatures
* File integrity checks
* TLS certificate verification

**Q63. Write Short Notes**

**a. Transport and Tunnel Mode in IPSec**

* **Transport Mode**: Encrypts **payload only**, used in end-to-end (host ↔ host)
* **Tunnel Mode**: Encrypts **entire packet**, used in gateway-to-gateway (e.g., VPN)

**b. S/MIME for Email Security**

* Secure/Multipurpose Internet Mail Extensions
* Provides **encryption + digital signatures**
* Uses **X.509 certificates**
* Built into most enterprise email clients

**c. TLS (Transport Layer Security)**

* Protocol ensuring **encrypted and secure communication**
* Common in HTTPS
* Uses **handshake** process:
  + ClientHello → ServerHello → Key Exchange → Session Key Created
  + Symmetric encryption used for speed, asymmetric used for exchange

**Example**:  
Visiting https://example.com triggers a TLS handshake to protect your data.

**Q64. Simulate Vernam Cipher for Encryption and Decryption**

**Vernam Cipher (One-Time Pad) Explanation:**

* **One-Time Pad (OTP)**: A symmetric-key encryption technique where the plaintext is combined with a key that is random, as long as the message, and used only once.
* **Key**: The key must be completely random, as long as the message, and never reused.
* **Encryption**:
  + Each bit of plaintext is XORed with the corresponding bit of the key.
  + Example: Plaintext = 110101, Key = 101010 → Ciphertext = 011111.
* **Decryption**: XOR the ciphertext with the same key to retrieve the original plaintext.

**Features:**

* **Perfect Security**: If key is random, as long as the message, and never reused, OTP is theoretically unbreakable.
* **Key Distribution**: The main challenge is secure key exchange.
* **Key Management**: The key must be the same length as the plaintext and kept secret.

**Example:**

* Plaintext: HELLO (in binary)
* Key: Random 5-letter sequence
* XOR the binary values for encryption and decryption.

You can use the **Vlabs simulation tool** to simulate this process and record:

* **Input Plaintext**: Example "HELLO" in binary
* **Key**: Random binary string
* **Resulting Ciphertext**: After XOR
* **Decryption**: XOR again with the same key to recover plaintext

**Q68. Explain Two Classical Ciphers**

**i. Playfair Cipher:**

* **Polygraphic Cipher**: Encrypts digraphs (pairs of letters).
* **Key**: 5x5 matrix of letters, where I and J are combined.
* **Encryption Example**:  
  Plaintext: HELLO  
  Key Matrix:
* H E L O A
* B C D F G
* I J K M N
* P Q R S T
* U V W X Y

The digraph "HE" → Look in matrix → Substitutes based on rules.

**ii. Hill Cipher:**

* **Polygraphic Cipher**: Encrypts blocks of text using matrix multiplication.
* **Key**: Square matrix of size n x n.
* **Encryption Example**:  
  Plaintext: ACT  
  Key: Matrix [[6, 24], [1, 16]]  
  Result is multiplied with plaintext vector to produce cipher.

**Q69. Short Notes**

**a. Transport and Tunnel Mode in IPSec**

* **Transport Mode**: Only the **payload** is encrypted. Used for **end-to-end** communication (host-to-host).
* **Tunnel Mode**: Encrypts **entire packet**, including the IP header. Used for **gateway-to-gateway** (site-to-site) communication.

**b. S/MIME for Email Security**

* **S/MIME (Secure/Multipurpose Internet Mail Extensions)**: Ensures email security using **X.509 certificates**.
* It provides **encryption** (privacy) and **digital signatures** (authenticity and integrity).
* Used widely in enterprise environments to secure email communications.

**c. TLS Explanation with Example**

* **TLS (Transport Layer Security)**: Ensures secure communication over a network (usually HTTP becomes HTTPS).
* **Example**:
  + **Client** connects to a web server, initiating a **TLS handshake**.
  + **Server** sends a certificate (public key), and both sides exchange encryption keys.
  + **Session key** established for encrypted communication.

**Q70. Explain AH and ESP Working in IPSec**

**AH (Authentication Header):**

* Provides **data integrity**, **authentication**, and **anti-replay** protection.
* **Not encrypting** data, just ensures that the data has not been altered.
* **Used for**: Verifying data authenticity between endpoints.

**ESP (Encapsulating Security Payload):**

* Provides **encryption** (confidentiality), **data integrity**, and **authentication**.
* **Encrypts** the payload and can include authentication.
* **Used for**: Securing the entire IP packet, including payload and header.

Both AH and ESP are part of IPSec, used to secure **IP traffic** through **VPNs**.