Experiment 1: Encryption-Decryption using Classical Cryptography (Caesar and Transposition)

Objective

- To write a C++ program to convert plain text into cipher text using the Caesar cipher and the Transposition cipher (both keyless and keyed columnar).

Background Theory

- **Cryptography:** The science of securing communication by transforming plaintext into ciphertext (encryption) and vice versa (decryption). It relies on mathematical principles and algorithms, often involving keys, to ensure confidentiality, integrity, authenticity, and non-repudiation.
- **Caesar Cipher:** A simple substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. For a shift of 'k', a letter 'L' becomes '(L + k) mod 26'.
  - **Security:** Very weak. Vulnerable to brute-force attack as there are only 25 possible non-trivial shifts for the English alphabet. Frequency analysis can also break it easily.
- **Transposition Cipher (Permutation Cipher):** Rearranges the positions of the plaintext letters according to a specific rule or key, without changing the letters themselves.
  - **Keyless Transposition:** Uses a predefined rule, like writing plaintext into a grid row-by-row and reading it column-by-column.
  - **Keyed Columnar Transposition:** Uses a keyword. Plaintext is written into a grid, and columns are read out in the order determined by the alphabetical order of the keyword's letters.
  - **Security:** More secure than simple substitution like Caesar, as it disrupts letter frequencies in adjacent positions. However, overall letter frequencies remain the same. Vulnerable to cryptanalysis (like frequency analysis or anagramming) if the pattern is simple or known. Security depends on the complexity of the rearrangement and key length.

Materials Used

- C++ Compiler and Development Environment
- Computer System
- Reference: www.geeksforgeeks.org

Procedure

Caesar Cipher Implementation (C++):

1. Include necessary headers (`<bits/stdc++.h>`).
2. Start an infinite loop (exits when user inputs "-1").
3. Prompt the user to enter the plaintext. Read the input string (`k`).
4. Check if the input is "-1"; if so, print a thank you message and break the loop.
5. Convert the plaintext to uppercase using `transform` .
6. Prompt the user for the desired shift value (`shift`). Read the integer input.
7. Create a copy of the plaintext string (`change`).
8. Iterate through each character of the `change` string:
   a. Calculate the 0-based position of the character: `pos = change[i] - 'A'` .
   b. Calculate the new shifted position using modular arithmetic, ensuring wrap-around and handling potential negative results: `newPos = (26 + ((pos + shift) % 26)) % 26` .
   c. Convert the `newPos` back to an uppercase character: `change[i] = newPos + 'A'` .
9. Print the resulting ciphertext (`change`).
10. **Brute-Force Decryption:**
    a. Print a header "Brute Force on it".
    b. Loop 26 times (for potential shifts `i` from 0 to 25).
    c. Inside this loop, iterate through each character of the original *ciphertext* (`change`).
    d. Calculate the potential plaintext position by applying the reverse shift `i` (or adding `i` in this specific code implementation, effectively trying all possible forward shifts on the ciphertext): `int newPos = (pos + i) % 26;` where `pos = change[j] - 65;` .
    e. Print the resulting character `(char)(newPos + 65)` .
    f. After iterating through all characters for shift `i` , print a newline to show the full potential plaintext for that shift.

Keyless Transposition Cipher Implementation (C++):

1. Include headers (`iostream` , `vector` , `string` , `cmath`).
2. Define the `keylessTranspositionCipher` function taking plaintext `string` .
3. Create an empty string `text` and copy the plaintext into it, removing spaces.
4. Define the number of columns (`numColumns = 4`).

5. Calculate the number of rows needed: `numRows = ceil(text.length() / numColumns)`.
6. Create a 2D grid (using `vector<string>`) of size `numRows` x `numColumns`, initialized with spaces.
7. Fill the grid row by row, character by character, using the `text` string.
8. Initialize an empty `ciphertext` string.
9. Read the grid column by column: Iterate through columns (outer loop), then rows (inner loop). Append non-space characters `grid[row][col]` to `ciphertext`.
10. Return the `ciphertext`.
11. In `main`, prompt for plaintext, call the function, and print the result.

Keyed Columnar Transposition Cipher Implementation (C++ - 5×5 Encrypt/Decrypt):

1. Include headers (`iostream`, `vector`, `string`, `algorithm`, `cmath`).
2. **Encryption (`columnarTranspositionCipher5x5Encrypt`):**
   a. Takes plaintext `string` and key `string`.
   b. Remove spaces from plaintext into `text`.
   c. Set `numColumns = 5`. Calculate `numRows`.
   d. Create a 2D grid (`vector<vector<char>>`).
   e. Fill the grid row by row with `text`.
   f. Create a `keyIndex` vector of pairs (`<int, char>`). Store `{original_index, key_char}` for each key character.
   g. Sort `keyIndex` based on the characters (`key[i]`).
   h. Initialize empty `ciphertext`.
   i. Iterate through the sorted `keyIndex`. For each pair, get the column index (`col = keyPair.first`).
   j. Read the characters down that column (`grid[row][col]`) and append non-space characters to `ciphertext`.
   k. Return `ciphertext`.
3. **Decryption (`columnarTranspositionCipher5x5Decrypt`):**
   a. Takes ciphertext `string` and key `string`.
   b. Set `numColumns = 5`. Calculate `numRows`.
   c. Create and sort `keyIndex` exactly as in encryption.
   d. Create an empty 2D grid (`vector<vector<char>>`).
   e. Initialize `index = 0` for ciphertext traversal.
   f. Iterate through the sorted `keyIndex`. For each pair, get the column index (`col = keyPair.first`).
   g. Fill that column (`grid[row][col]`) using characters from the `ciphertext` sequentially until the column is full or ciphertext ends. Increment `index`.
   h. Initialize empty `plaintext`.

i. Read the grid row by row: Iterate through rows, then columns. Append non-space characters `grid[row][col]` to `plaintext`.
j. Return `plaintext`.

4. **Main Function:**
a. Prompt for plaintext and key (max length 5). Check key length.
b. Call encryption function, print ciphertext.
c. Call decryption function, print decrypted text.

Observations

- **Caesar Cipher Example:**
  - Input Plaintext: `Yashank`
  - Input Shift: `5`
  - Ciphertext Output: `DFXMFSP`
  - Brute force output shows all 26 possible shifts applied to `DFXMFSP`, one of which reveals the original plaintext `YASHANK`.
- **Keyless Transposition Example:**
  - Input Plaintext: `yashank` (spaces removed)
  - `numColumns = 4`
  - Ciphertext Output: `yaanskh` (Grid: y a s h | a n k ' ')
- **Keyed Columnar Transposition Example:**
  - Input Plaintext: `ENEMYATTACKSTONIGHT`
  - Input Key: `hacks` (length 5)
  - Ciphertext Output: `NTSGTTHEAKIMAQTYCN`
  - Decrypted Text Output: `ENEMYATTACKSTONIGHT`

Results/Calculations

- The primary results are the generated ciphertexts and decrypted plaintexts.
- Calculations involve:
  - Caesar: Modular arithmetic `(char_pos + shift) % 26`.
  - Transposition: Grid dimension calculation (`ceil(len / cols)`), row/column indexing, sorting based on key characters.

Discussion/Analysis

- The Caesar cipher implementation correctly shifts letters and demonstrates the ease of brute-force attack by displaying all possible decryptions.
- The Keyless Transposition implementation correctly rearranges letters based on a fixed grid size (4 columns). Its security is minimal as the pattern is fixed.

- The Keyed Columnar Transposition implementation uses a 5-letter key to determine the column order, providing better security than the keyless version. The decryption function successfully reverses the process.
- Both transposition ciphers preserve original letter frequencies, making them potentially vulnerable to frequency analysis, especially with longer texts.
- The implementations primarily handle uppercase letters and remove spaces (in transposition). They do not handle numbers or punctuation.

Conclusion

- Learned about classical cryptography principles (substitution and transposition).
- Successfully implemented Caesar cipher (including brute-force), keyless transposition, and keyed columnar transposition in C++.
- Observed the outputs and the relative simplicity/vulnerability of these classical methods.

Experiment 2: Application of RSA

Objective

- To understand the theoretical framework of RSA encryption.
- To explore and apply RSA principles using OpenSSL command-line tools for key generation, encryption, decryption, hashing, and digital signatures.

Background Theory

- **RSA (Rivest-Shamir-Adleman):** A fundamental asymmetric (public-key) cryptographic algorithm. Its security relies on the computational difficulty of factoring large prime numbers.
- **Key Generation Process:**
  1. **Prime Selection:** Choose two large, distinct, random prime numbers, `p` and `q` (typically 1024-4096 bits).
  2. **Modulus Computation:** Calculate `n = p * q`. `n` is the public modulus.
  3. **Euler's Totient Function:** Compute `ϕ(n) = (p−1) * (q−1)`. This counts integers less than `n` that are coprime to `n`.
  4. **Public Exponent Selection:** Choose an integer `e` such that `1 < e < ϕ(n)` and `e` is coprime to `ϕ(n)` (i.e., `gcd(e, ϕ(n)) = 1`). Common choice is 65537.
  5. **Private Exponent Computation:** Calculate `d` such that `(d * e) mod ϕ(n) = 1`. `d` is the modular multiplicative inverse of `e` modulo `ϕ(n)`.
- **Keys:**

    - Public Key: `(n, e)` - Shared openly.
    - Private Key: `(n, d)` - Kept secret.
- **Encryption Mechanism:** `C = M^e mod n` (where M is the plaintext message, C is the ciphertext). Performed using the public key. Computationally easy.
- **Decryption Mechanism:** `M = C^d mod n` (where C is the ciphertext, M is the original message). Performed using the private key `d`.
- **Security Principles:** Breaking RSA is equivalent to factoring `n` into `p` and `q`, which is computationally infeasible for large `n`. Security increases exponentially with key size.
- **Proof of Correctness:** Based on Euler's theorem, `(M^e)^d ≡ M (mod n)`.
- **Cryptographic Significance:** Revolutionized secure communication by enabling:
    - Asymmetric Encryption (Confidentiality)
    - Digital Signatures (Authenticity, Integrity, Non-repudiation)
    - Secure Key Exchange
    - Public Key Infrastructure (PKI) foundation.
- **Practical Limitations:** Slower than symmetric encryption, requires large keys, potentially vulnerable to quantum computing, factorization advances, and side-channel attacks. Does not inherently provide forward secrecy.

Materials Used

- OpenSSL command-line tool
- Linux/Unix-like Terminal or Windows Command Prompt with OpenSSL installed
- Text editor (for creating sample files like `plaintext.txt`, `message.txt`)
- Referenced Websites: sandilands.info, 0xshakhawat.medium.com

Procedure (Using OpenSSL Commands)

1. **Generate RSA Private Key:** Create a private key file.

```
openssl genpkey -algorithm RSA -out private_key.pem
```

2. **Extract Public Key:** Derive the public key from the private key.

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

3. **File Encryption:** Encrypt a sample file (`plaintext.txt` or `message.txt`) using the recipient's public key.

```
# Create a sample file first: echo "This is a secret message" > message.txt
openssl pkeyutl -encrypt -in message.txt -out encrypted.bin -pubin -inkey
public_key.pem
# (Note: rsautl is older, pkeyutl is preferred)
```

4. **File Decryption:** Decrypt the received file (`encrypted.bin`) using the recipient's private key.

```
openssl pkeyutl -decrypt -in encrypted.bin -out decrypted.txt -inkey
private_key.pem
```

5. **Generate Hash:** Compute a cryptographic hash (MD5 shown in output, SHA-256 mentioned in theory) of a file (`plaintext1.txt`).

```
# Create sample file: echo "Some content" > plaintext1.txt
openssl dgst -md5 plaintext1.txt
# For SHA256: openssl dgst -sha256 plaintext1.txt
```

6. **Generate Self-Signed Certificate (Illustrative):** Create a private key and then a self-signed certificate (often used for testing).

```
# Generate a key (if not already done)
# openssl genpkey -algorithm RSA -out private.key
# Generate a Certificate Signing Request (CSR)
# openssl req -new -key private.key -out request.csr
# Generate self-signed certificate from CSR and key
openssl x509 -req -days 365 -in request.csr -signkey private.key -out
certificate.pem
```

(Screenshots show certificate generation steps)

7. **Create Digital Signature:** Sign a file (`original.txt`) using a private key.

```
# Create sample file: echo "Data to be signed" > original.txt
openssl dgst -sha256 -sign privatekey.pem -out signature.bin original.txt
```

(Screenshots show this command)

8. **Verify Digital Signature:** Verify the signature (`signature.bin`) against the original file (`original.txt`) using the corresponding public key.

```
openssl dgst -sha256 -verify publickey.pem -signature signature.bin
original.txt
```

(Screenshots show this command, likely resulting in "Verified OK")

Observations

- OpenSSL commands execute successfully, generating key files (`.pem`), encrypted binary files (`.bin`), decrypted text files (`.txt`), and signature files (`.bin`).
- Key generation produces characteristic PEM-encoded text output.
- Hash function (`dgst -md5`) produces a hexadecimal hash value (e.g., `885d82c3c05eb98aabb5d5a93c2ed4fb` for `plaintext1.txt`).
- Certificate generation results in a `.pem` file containing the certificate details (Issuer, Subject, Validity period shown in screenshot: Issued to/by `yashank` , Valid from 30-01-2025 to 30-01-2026).
- Digital signature commands require password entry for the private key. Verification command outputs status (e.g., "Verified OK").
- Virtual Lab shows RSA encryption/decryption of text ("test" → hex ciphertext → "test") and key parameters (Modulus, Public/Private exponents, primes p & q, etc.).

Results/Calculations

- **Keys:** Generated public/private RSA key pairs.
- **Confidentiality:** Successfully encrypted and decrypted files using the key pair.
- **Integrity/Hashing:** Calculated MD5 hash of a file.
- **Authentication/Signatures:** Successfully created and verified a digital signature.
- **Post-Lab Calculation:**
  - Given: `p=7` , `q=11` , `e=13`
  - `n = p * q = 7 * 11 = 77`
  - `ϕ(n) = (p-1) * (q-1) = (7-1) * (11-1) = 6 * 10 = 60`
  - Find `d` such that `(d * 13) mod 60 = 1` .
  - Using Extended Euclidean Algorithm:
    - `60 = 4 * 13 + 8`
    - `13 = 1 * 8 + 5`
    - `8 = 1 * 5 + 3`
    - `5 = 1 * 3 + 2`
    - `3 = 1 * 2 + 1`
    - `1 = 3 - 1 * 2`
    - `1 = 3 - 1 * (5 - 1 * 3) = 2 * 3 - 1 * 5`

        - `1 = 2 * (8 - 1 * 5) - 1 * 5 = 2 * 8 - 3 * 5`
        - `1 = 2 * 8 - 3 * (13 - 1 * 8) = 5 * 8 - 3 * 13`
        - `1 = 5 * (60 - 4 * 13) - 3 * 13 = 5 * 60 - 20 * 13 - 3 * 13 = 5 * 60 - 23 * 13`
    - So, `-23 * 13 mod 60 = 1`.
    - We need positive `d`: `d = -23 mod 60 = -23 + 60 = 37`.
    - Therefore, `d = 37`.

Discussion/Analysis

- RSA provides confidentiality through encryption/decryption using the public/private key pair, demonstrated by the file encryption/decryption steps.
- RSA provides authenticity, integrity, and non-repudiation through digital signatures, demonstrated by signing and verification steps.
- Hashing provides integrity checks.
- Certificates link public keys to identities, crucial for PKI.
- **Cryptanalysis:** RSA is vulnerable if keys are too small or primes are weak. Attacks include:
    - *Factorization:* Directly attacking `n` (GNFS). Defense: Large keys (2048+ bits).
    - *Side-Channel:* Timing, power, EM analysis. Defense: Constant-time implementations.
    - *Padding Oracle:* Exploiting padding errors (e.g., Bleichenbacher's attack on PKCS#1 v1.5). Defense: Secure padding (OAEP).
    - *Chosen Plaintext/Ciphertext (CPA/CCA):* Exploiting mathematical properties, especially without proper padding. Defense: Randomised padding (OAEP).
- **Drawbacks & Solutions:**
    - *Slow Computation:* Use Hybrid Encryption (AES for data, RSA for key).
    - *Large Key Size:* Use ECC for similar security with smaller keys.
    - *Factorization Vulnerability:* Use large, random primes; increase key size periodically.
    - *Side-Channel Susceptibility:* Use constant-time code, hardware protection.
    - *No Forward Secrecy:* Compromised key reveals past messages. Use ephemeral key exchange (Diffie-Hellman) for session keys.

Conclusion

- Learned the theoretical basis of RSA encryption, including key generation, encryption, and decryption processes.

- Practically applied RSA concepts using OpenSSL commands to generate keys, encrypt/decrypt data, compute hashes, and create/verify digital signatures.
- Understood the security principles, common attacks, limitations, and defenses related to RSA. RSA remains a cornerstone of modern cryptography despite its limitations.

Experiment 3: Implementation of CAPTCHA for Security of Systems

Objective

- To understand the purpose and function of CAPTCHA.
- To implement and test various types of CAPTCHA mechanisms (Text, Math, Distorted Text, Image Grid) using HTML, CSS, and JavaScript.
- To analyze the security aspects and usability trade-offs of different CAPTCHA types.

Background Theory

- **CAPTCHA:** Completely Automated Public Turing test to tell Computers and Humans Apart. A challenge-response test used in computing to determine whether the user is human.
- **Purpose:** Primarily used for bot prevention. Protects websites from automated attacks like spam submissions, brute-force login attempts, web scraping, and denial-of-service attacks.
- **Inspiration:** Based on the Turing Test concept, which tests a machine's ability to exhibit intelligent behavior indistinguishable from a human.
- **Types of CAPTCHA:**
  - **Text-based:** User types distorted letters/digits from an image. Most common, but increasingly vulnerable to OCR.
  - **Mathematical:** User solves a simple math problem (e.g., 3+4=?). Easy for humans, but very easy for bots.
  - **Image-based:** User identifies objects or patterns in images (e.g., "select all traffic lights"). Relies on image recognition.
  - **Grid-based:** A variation of image-based, user selects specific squares in a grid based on a prompt (e.g., "select squares containing cars").
  - **Audio:** An audio clip plays distorted numbers/letters for visually impaired users.
  - **reCAPTCHA (Google):** More advanced, often uses behavioral analysis (mouse movements, interaction timing) and risk analysis. Can be invisible or present

challenges like image selection.

- **Security Aspects:** Enhances security by ensuring only humans can perform certain actions (submit forms, login, post comments). Acts as a filter against automated scripts.
- **Challenges & Weaknesses:**
  - **Evolving Bots:** Bots using AI/ML are becoming better at solving CAPTCHAs.
  - **Usability:** Can be frustrating and time-consuming for users.
  - **Accessibility:** Can be difficult or impossible for users with disabilities (visual, auditory, cognitive).
  - **Security vs. Usability Trade-off:** More difficult CAPTCHAs are more secure but less user-friendly.

Materials Used

- Web Browser (Chrome, Firefox, etc.)
- Text Editor (VS Code, Sublime Text, etc.)
- HTML, CSS, JavaScript files (provided/created for the experiment)
- Image files for the Grid Image CAPTCHA
- Referenced Websites: YouTube, geeksforgeeks

Procedure

1. **Setup:** Create an `index.html` page with links to separate HTML pages for each CAPTCHA type (`text.html`, `maths.html`, `textimg.html`, `gridimg.html`). Apply common CSS styling for layout, containers, buttons.
2. **Text CAPTCHA (`text.html`):**
   a. **HTML:** Create a `div.captcha-container` containing a `p#text-captcha` to display text, an `<input type="text" id="text-input">`, a "Generate Captcha" button (`onclick="generateTextCaptcha()"`), a "Submit" button (`onclick="verifyTextCaptcha()"`), and a `div#message` for results.
   b. **JavaScript:**
   `generateTextCaptcha()`: Defines a character pool (A-Z, a-z, 0-9). Generates a 6-character random string. Sets the `textContent` of `p#text-captcha` to this string. Stores the string in `p#text-captcha.dataset.captcha`. Clears the message div.
   `verifyTextCaptcha()`: Gets the stored captcha from `dataset.captcha` and the user's input value. Compares them. Sets the `textContent` and color (`green` / `red`) of `div#message` accordingly.
3. **Mathematical CAPTCHA (`maths.html`):**
   a. **HTML:** `div.captcha-container` with `p#math-captcha` displaying "3 + 4 = ?", an `<input type="number" id="math-input">`, a "Submit" button

(`onclick="verifyMathCaptcha()"`), and `div#message` .
b. **JavaScript:**
* `verifyMathCaptcha()` : Defines `correctAnswer = 7` . Parses the user's input value as an integer. Compares input to `correctAnswer` . Sets `div#message` text and color. (Note: Example uses a static problem).

4. **Distorted Text (Canvas) CAPTCHA (`textimg.html`):**
a. **HTML:** `div.captcha-container` with `<canvas id="captcha-canvas" width="200" height="60">` , "Refresh Captcha" button (`onclick="drawCanvasCaptcha()"`), "Submit" button (`onclick="verifyCanvasCaptcha()"`), `<input type="text" id="canvas-input">` , and `div#message` .
b. **JavaScript:**
*`drawCanvasCaptcha()` : Gets canvas and 2D context. Defines character pool. Generates 6-char random string `captchaText` . Stores it in `canvas.dataset.captcha` . Clears canvas (`clearRect`), fills with background (`#ddd`). Sets font (`40px Arial`), color (`blue`). Saves context state (`save()`). Loops 6 times: applies a small random rotation (`ctx.rotate`), draws the i-th character (`fillText`) at `(i* 30 + 10, 45) .` `Restores context state (` restore() `). Called once initially. *`
verifyCanvasCaptcha() `: Gets user input and expected captcha from` dataset.captcha `. Compares them. Sets` div#message` text and color.

5. **Grid Image CAPTCHA (`gridimg.html`):**
a. **HTML:** `div.captcha-container` with instructions (`<p>Select the pasta images</p>`), `<div id="grid-container" class="grid-captcha">` , "Refresh Grid" button (`onclick="generateGridCaptcha()"`), "Submit" button (`onclick="verifyGridCaptcha()"`), and `div#message` .
b. **CSS:** Styles for `.grid-captcha` (flex display), `img` (size, cursor, border), and `img.selected` (e.g., `border: 2px solid blue`).
c. **JavaScript:**
`generateGridCaptcha()` : Gets `gridContainer` . Clears `innerHTML` . Defines an `images` array of objects `{src: "path/to/img.png", isPasta: true/false}` . Shuffles the array (`images.sort(() => Math.random() - 0.5)`). Loops through `shuffledImages` : creates `<img>` , sets `src` , sets `img.dataset.isPasta` . Adds a click event listener to the image that toggles the `selected` class on the image. Appends `img` to `gridContainer` . Called once initially.
`verifyGridCaptcha()` : Selects all `img.selected` elements. If none selected, show error message and return. Set `valid = true` . Loops through `selectedImages` : if `img.dataset.isPasta` is not `"true"` , set `valid = false` . Set `div#message` text and color based on `valid` .

Observations

- The main `index.html` page displayed links to the four CAPTCHA types.

- **Text CAPTCHA:** Generated random strings (e.g., `RyqBuD`). Correct input resulted in "Captcha verified!" (green). Incorrect input resulted in "Wrong captcha!" (red).
- **Math CAPTCHA:** Displayed "3 + 4 = ?". Inputting `7` resulted in "Captcha verified!". Inputting other numbers (e.g., `8`) resulted in "Incorrect answer!".
- **Distorted Text CAPTCHA:** Displayed a canvas with distorted text (e.g., `Ur OF Kz`). Correct input resulted in "Captcha verified!". Incorrect input resulted in "Incorrect captcha!". Refresh button generated new text.
- **Grid Image CAPTCHA:** Displayed a grid of images (pasta, cars, trees). Clicking images added a blue border. Clicking "Submit" after selecting only pasta images resulted in "Captcha verified!". Selecting any non-pasta image resulted in "Incorrect selection!". Clicking "Refresh Grid" changed the image arrangement.

Results/Calculations

- Successfully implemented four different CAPTCHA interfaces.
- Verification logic correctly distinguished between correct and incorrect user inputs based on the generated challenge.
- Calculations involved random string/number generation, simple arithmetic, array shuffling, DOM manipulation, and string comparison.

Discussion/Analysis

- Each CAPTCHA type presents a different challenge to bots.
- Text CAPTCHA is standard but potentially weak against modern OCR.
- Math CAPTCHA is very weak against automation.
- Distorted Text (Canvas) increases difficulty for basic OCR but might also hinder human users.
- Image Grid CAPTCHA relies on object recognition, generally harder for bots but not impossible for AI, and poses accessibility challenges.
- **reCAPTCHA vs. CAPTCHA:** Traditional CAPTCHA uses direct challenges. reCAPTCHA incorporates behavioral analysis and AI for risk assessment, often providing a smoother user experience (invisible CAPTCHA) but relying on Google's infrastructure.
- **Limitations:** User frustration, accessibility issues, advancing bot technology, difficulty on mobile devices.
- **Alternatives:** Behavioral analysis (like reCAPTCHA v3), biometric authentication, honeypot fields, Two-Factor Authentication (2FA), email verification links.

Conclusion

- Successfully learned about and implemented various types of CAPTCHA mechanisms (Text, Math, Distorted Text, Image Grid).
- Understood their role in enhancing web security by differentiating humans from bots.
- Recognized the trade-offs between security, usability, and accessibility, and the need for continuous evolution in bot detection techniques.

Experiment 4: Email Security using PGP (Mailvelope)

Objective

- To perform various activities related to email security using PGP, specifically leveraging the Mailvelope browser extension.
- To understand and practice key generation, key management, email encryption/decryption, digital signing/verification, and encrypted attachments.

Background Theory

- **Public Key Cryptography:** The foundation of PGP/Mailvelope. Uses asymmetric key pairs: a public key (shared openly for encryption and signature verification) and a private key (kept secret for decryption and signing).
- **OpenPGP:** (Pretty Good Privacy) A standard (RFC 4880) defining protocols for encrypting and signing data, commonly used for email security. Ensures:
  - **Confidentiality:** Only the intended recipient (with the private key) can decrypt the message encrypted with their public key.
  - **Authentication:** Digital signatures verify the sender's identity.
  - **Integrity:** Digital signatures ensure the message hasn't been tampered with.
  - **Non-repudiation:** The sender cannot deny sending a signed message.
- **Digital Signatures:** Created by hashing the message and encrypting the hash with the sender's private key. Verified by decrypting the signature hash with the sender's public key and comparing it to a hash of the received message.
- **Mailvelope:** A browser extension that integrates OpenPGP functionality into webmail interfaces (like Gmail, Outlook.com), facilitating end-to-end encryption.
- **End-to-End Encryption (E2EE):** The message is encrypted by the sender and decrypted only by the recipient. Intermediate servers (like the email provider) cannot read the content, protecting against surveillance and Man-in-the-Middle (MITM) attacks.
- **Key Management:** Crucial for PGP. Involves generating key pairs, securely storing the private key (usually password-protected), exchanging public keys with contacts

(via email, key servers, manually), importing contact keys, and potentially revoking compromised keys.

Materials Used

- Web Browser (e.g., Chrome, Firefox)
- Mailvelope Browser Extension
- Webmail Account (e.g., Gmail)
- Internet Connection

Procedure (Using Mailvelope)

1. **Installation:** Install the Mailvelope browser extension.
2. **Key Generation:**
   a. Open Mailvelope options → Key Management.
   b. Click "Generate" to create a new key pair.
   c. Enter Name (e.g., Yashank), Email, optionally advanced options.
   d. Set a strong password to protect the private key.
   e. The key pair is generated and listed (e.g., Key ID 4F16616ABF0786689166EF, RSA 4096-bit).
3. **Key Management:**
   a. View key details (Status: valid, Created date, Expires date, Algorithm, Length, Fingerprint).
   b. Import public keys from contacts (e.g., import Minav's key - OC9B9A568D5090186D8). Keys can be imported from files or key servers.
   c. Export *your* public key to share with others.
   d. Optionally backup/export your key pair.
   e. Options to Revoke or Remove keys.
4. **Sending a Signed Email:**
   a. Compose a new email in the webmail interface. Mailvelope adds its editor/icon.
   b. Write the message (e.g., "this is a message to test mailvelope extension").
   c. Click the Mailvelope "Sign" option (often combined with Encrypt/Sign icon, then choose Sign Only).
   d. Mailvelope prompts for the private key password. Enter it.
   e. Send the email. The message body itself is not encrypted, but PGP signature data is added.
5. **Verifying a Signed Email:**
   a. The recipient (with Mailvelope installed and the sender's public key imported) opens the email.

b. Mailvelope automatically verifies the signature and displays a status indicator (e.g., green checkmark, "Signed by Yashank..").

6. **Sending an Encrypted Email:**
   a. Compose a new email. Enter recipient(s) whose public keys are known to Mailvelope.
   b. Write the message.
   c. Click the Mailvelope "Encrypt" option.
   d. Mailvelope encrypts the message using the recipient(s)' public key(s). Optionally sign as well (requires password).
   e. Send the email. The body will appear as PGP encrypted block.
7. **Decrypting an Encrypted Email:**
   a. Open the received encrypted email.
   b. Mailvelope detects the encrypted block and prompts for the private key password.
   c. Enter the password.
   d. Mailvelope decrypts the message and displays the original plaintext. (Screenshot shows prompt for Minav's key password).
8. **Sending Encrypted Attachments:**
   a. Compose email, enter recipient, write message.
   b. Attach the file (e.g., `Yashank.pdf`).
   c. Select the Mailvelope "Encrypt" option (this typically encrypts both message and attachment).
   d. Enter password if also signing. Send.
9. **Decrypting Encrypted Attachments:**
   a. Open the received email.
   b. Decrypt the message body by entering the private key password.
   c. Mailvelope provides access to the decrypted attachment (e.g., a download link or displays it). The screenshot shows the PDF icon after decryption.

Observations

- Mailvelope integrates seamlessly into the webmail compose window.
- Key Management interface allows generating, importing, exporting, and viewing keys with details like Key ID, fingerprint, algorithm (RSA 4096), status, and associated User IDs.
- Password prompts appear for actions requiring the private key (signing, decryption).
- Clear visual indicators show whether a message is signed, encrypted, or successfully decrypted/verified.
- Encrypted messages appear as blocks of PGP data (`———BEGIN PGP MESSAGE———` `-...`).
- Encrypted attachments are handled alongside the message encryption.

Results/Calculations

- Successfully generated RSA 4096-bit key pairs.
- Successfully sent and verified digitally signed emails.
- Successfully sent end-to-end encrypted emails and decrypted them.
- Successfully sent and accessed encrypted attachments.
- Internal cryptographic operations (RSA, symmetric encryption like AES for message body, hashing for signatures) are handled by Mailvelope/OpenPGP libraries.

Discussion/Analysis

- Mailvelope provides a practical way to implement strong end-to-end email security using the OpenPGP standard directly within webmail.
- It effectively ensures confidentiality (encryption) and authenticity/integrity (digital signatures).
- The security relies heavily on proper key management: protecting the private key with a strong password and verifying the authenticity of contacts' public keys.
- Compared to simple TLS encryption (which secures the connection *between* servers), PGP/Mailvelope provides E2EE, securing the message content *at rest* and *in transit* from unauthorized access, including by the email provider.
- Usability remains a factor; users need to understand key management concepts.

Conclusion

- Through the practical implementation using Mailvelope, learned the significance and process of securing email communication using asymmetric cryptography (OpenPGP).
- Successfully utilized Mailvelope for generating key pairs, exchanging public keys (implicitly), encrypting/decrypting messages, applying/verifying digital signatures, and handling encrypted attachments.
- Demonstrated the protection of confidentiality, authenticity, and integrity in emails via E2EE.

Experiment 5: Firewall Configuration using iptables

Title

- Firewall Configuration using iptables *(Inferred, as title was blank)*

Objective

- *(Inferred)* To learn and apply basic `iptables` commands for Linux firewall configuration.
- To understand firewall concepts like tables, chains, rules, targets, and default policies.
- To configure rules for common scenarios: setting default policies, allowing essential traffic (loopback, established connections), blocking specific IPs/ports, allowing specific services (HTTP/S, email ports), implementing NAT, and managing rule persistence.

Background Theory

- **Firewall:** A network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules. Establishes a barrier between a trusted internal network and untrusted external networks.
- **iptables:** A user-space command-line utility for configuring the Linux kernel firewall (Netfilter module). It allows system administrators to define rules for filtering network packets.
- **Tables:** Organize rules based on their purpose. Main tables:
  - `filter`: Default table, used for packet filtering (allow/block). Contains INPUT, FORWARD, OUTPUT chains.
  - `nat`: Network Address Translation. Used for routing packets to networks they cannot access directly. Contains PREROUTING, OUTPUT, POSTROUTING chains.
  - `mangle`: Modifying packet headers. Contains PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING chains.
- **Chains:** A sequence of rules applied to packets based on their path through the system.
  - `INPUT`: Packets destined for the local system.
  - `OUTPUT`: Packets originating from the local system.
  - `FORWARD`: Packets being routed through the local system.
  - `PREROUTING` (nat, mangle): Packets as soon as they arrive at the network interface.
  - `POSTROUTING` (nat, mangle): Packets just before they leave the network interface.
- **Rules:** Specify criteria (matches) for packets and an action (target) to take if a packet matches.

    - *Matches:* Source/Destination IP (`-s`, `-d`), Protocol (`-p tcp/udp/icmp`), Source/Destination Port (`--sport`, `--dport`), Interface (`-i`, `-o`), Connection State (`-m state --state NEW,ESTABLISHED,RELATED,INVALID`).
    - *Targets:* `ACCEPT` (allow), `DROP` (discard silently), `REJECT` (discard with error notification), `LOG` (log packet details), `MASQUERADE` (NAT for dynamic IPs).
- **Default Policy:** The action taken if a packet doesn't match any rule in a chain (`-P INPUT DROP`). A default `DROP` policy is generally more secure.

Materials Used

- Linux Operating System (e.g., Ubuntu, CentOS) with `iptables` installed
- Terminal / Command Line Interface
- Root or `sudo` privileges

Procedure (Executed `iptables` Commands)

(Based on screenshots provided)

1. **Set Default Policies:** Block incoming/forwarding by default, allow outgoing.

```
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT
```

2. **Allow Loopback Traffic:** Permit traffic on the loopback interface (`lo`).

```
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A OUTPUT -o lo -j ACCEPT
```

3. **Allow Established/Related Connections:** Allow incoming traffic that is part of an existing connection initiated from the system or related to it (important for return traffic).

```
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

4. **Allow Internal Network to Access External (NAT):** Configure Network Address Translation (Masquerading) for an internal network (e.g., 192.168.1.0/24) using an external interface (e.g., `eth0`).

```
sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j MASQUERADE
```

5. **Drop Invalid Packets:** Drop packets that don't conform to expected connection states.

```
sudo iptables -A INPUT -m state --state INVALID -j DROP
sudo iptables -A FORWARD -m state --state INVALID -j DROP
```

6. **Block Specific Outgoing IP Addresses:** Prevent connections *to* specific destination IPs.

```
sudo iptables -A OUTPUT -d 142.250.183.206 -j DROP
sudo iptables -A OUTPUT -d 142.250.182.14 -j DROP
```

7. **Allow Incoming HTTP/HTTPS:** Allow connections *to* web server ports 80 and 443.

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

8. **Block Outgoing SMTP Mail:** Prevent sending email via standard SMTP ports (using REJECT sends an error back).

```
sudo iptables -A OUTPUT -p tcp --dport 25 -j REJECT
sudo iptables -A OUTPUT -p tcp --dport 465 -j REJECT # SMTPS
sudo iptables -A OUTPUT -p tcp --dport 587 -j REJECT # Submission
```

9. **Allow Incoming Email Ports:** Allow connections *to* common email ports (IMAP, POP3, secure versions).

```
sudo iptables -A INPUT -p tcp --dport 143 -j ACCEPT  # IMAP
sudo iptables -A INPUT -p tcp --dport 993 -j ACCEPT  # IMAPS
sudo iptables -A INPUT -p tcp --dport 110 -j ACCEPT  # POP3
sudo iptables -A INPUT -p tcp --dport 995 -j ACCEPT  # POP3S
```

10. **Allow Incoming SSH (Example):** Allow connections *to* SSH port 22.

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

11. **Manage Rules:**

- **List:** View current rules with details and line numbers.

```
sudo iptables -L -v -n --line-numbers
```

- **Delete:** Remove a rule by its chain and line number (e.g., rule 3 in INPUT).

```
sudo iptables -D INPUT 3
```

- **Save:** Persist the current ruleset to a file.

```
sudo iptables-save > ~/iptables-backup.rules
```

- **Restore (Implied):** Load rules from a file (e.g., on boot).

```
sudo iptables-restore < ~/iptables-backup.rules
```

12. **Restrict Access by Time:** Block outgoing access to `google.com` during specific hours (10:00 to 18:00).

```
sudo iptables -A OUTPUT -p tcp -d google.com -m time --timestart 10:00 --
timestop 18:00 -j REJECT
```

Observations

- Commands require `sudo` privileges.
- Most commands don't produce output unless successful (or if there's an error).
- `iptables -L -v -n --line-numbers` displays the ruleset in a structured format, showing chains, policies, rules with criteria (protocol, source, destination, ports, state) and targets.
- The order of rules within a chain matters, as the first matching rule determines the packet's fate.

Results/Calculations

- A functional firewall configuration based on the executed rules.
- Traffic flow is controlled according to the defined ACCEPT, DROP, REJECT, and MASQUERADE rules.
- Specific services are allowed (HTTP/S, SSH, Email) or blocked (SMTP out).

- Specific IPs are blocked.
- Rules can be listed, deleted, and saved for persistence.

Discussion/Analysis

- Implementing a default DROP policy for INPUT and FORWARD chains significantly enhances security by requiring explicit ALLOW rules for desired traffic.
- Allowing loopback (`lo`) traffic is essential for many local services.
- Allowing `ESTABLISHED,RELATED` stateful connections is crucial for allowing return traffic for connections initiated by the system, without needing overly broad INPUT rules.
- NAT (`MASQUERADE`) enables internal devices to share a single public IP address.
- Blocking specific IPs or ports is a direct way to prevent communication with known malicious hosts or unwanted services.
- Time-based rules allow for granular control based on time of day.
- `iptables-save` and `iptables-restore` are necessary because `iptables` rules are not persistent across reboots by default (persistence mechanisms vary by distribution, e.g., `iptables-persistent` package).

Conclusion

- Successfully configured a Linux firewall using `iptables` commands.
- Implemented fundamental firewalling techniques, including setting default policies, managing stateful connections, allowing/blocking specific protocols, ports, and IP addresses, configuring NAT, and ensuring rule persistence.
- Gained practical understanding of `iptables` syntax and firewall concepts.

Experiment 6: SQL Injection Using Burpsuite/Without Burpsuite

Objective

- To understand the concept and impact of SQL Injection (SQLi) vulnerabilities.
- To perform standard (Union-based) and Blind SQL Injection attacks manually against a vulnerable web application (DVWA).
- To utilize Burp Suite (Proxy and Intruder) to aid in detecting and exploiting Blind SQL Injection.

Background Theory

- **SQL Injection (SQLi):** A code injection technique where malicious SQL statements are inserted into data entry fields (like forms or URL parameters) of a web application. The injected code is then executed by the backend database, potentially leading to unauthorized data access, modification, or deletion.
- **Impact:** Expose sensitive data, compromise user privacy (credentials, personal info), gain administrative database/system access, alter/delete data, denial of service. One of the most common and critical web vulnerabilities.
- **DVWA (Damn Vulnerable Web Application):** An open-source PHP/MySQL web application intentionally designed with security vulnerabilities (like SQLi, XSS, CSRF) for security professionals and students to practice penetration testing techniques in a safe, legal environment. It offers different security levels (Low, Medium, High, Impossible) to demonstrate various defenses.
- **Burp Suite:** An integrated platform for web application security testing. Key components used:
  - **Proxy:** Intercepts HTTP/S traffic between the browser and the target server, allowing inspection and modification of requests/responses.
  - **Intruder:** Automates customized attacks by sending numerous modified HTTP requests. Used here to test various SQLi payloads rapidly.
- **Types of SQLi:**
  - **In-Band (Classic):** Uses the same communication channel to launch the attack and gather results. (e.g., Union-based: uses `UNION` operator to combine results from a crafted query with the original query).
  - **Blind:** Does not retrieve data directly in the response. Attacker infers information by observing the application's behavior (e.g., differences in response time or content for true/false conditions).

Materials Used

- Web Browser
- DVWA Instance (likely running locally via XAMPP/Docker or on Kali)
- Burp Suite (Community Edition sufficient)
- Kali Linux (Optional, includes DVWA/Burp)
- Referenced Websites: kali.org, portswigger.net

Procedure

1. **Setup:**
   a. Start DVWA (ensure database is running).

b. Access DVWA via browser, log in (default: admin/password).
c. Navigate to "DVWA Security" and set the level to "Low".
d. Configure browser to use Burp Suite Proxy (typically `127.0.0.1:8080`). Ensure Burp's Intercept is initially off.

2. **Standard SQL Injection (Manual - DVWA Low Security):**
   a. Navigate to the "SQL Injection" page in DVWA.
   b. Enter a normal User ID: `1`. Click Submit. Observe result (shows admin's details).
   c. Enter a basic tautology: `1' OR '1'='1`. Click Submit. Observe result (shows details for all users: admin, Gordon, Hack, Pablo, Bob). This confirms basic SQLi.
   d. Perform Union-based injection to extract information (assuming 2 columns based on common DVWA structure):
   Database Version: `1' union select null,@@version #` *→ Observe version (e.g.,* `10.4.32-MariaDB`*).*
   Database Name: `1' union select null,database() #` → Observe name (`dvwa`).
   Database User: `1' union select null,user() #` *→ Observe user (* `dvwa@localhost`*).*
   Hostname: `1' union select null,@@hostname #` → Observe hostname (e.g., `KJSCE0209-18`).
   * Data Directory: `1' union select null,@@datadir #` → Observe path (`c:\xampp\mysql\data\`).

3. **Blind SQL Injection (Manual - DVWA Low Security):**
   a. Navigate to the "SQL Injection (Blind)" page in DVWA.
   b. Enter User ID that exists, appending a TRUE condition: `1' AND '1'='1`. Click Submit. Observe response: "User ID exists in the database."
   c. Enter User ID that exists, appending a FALSE condition: `1' AND '1'='2`. Click Submit. Observe response: "User ID is MISSING from the database."
   d. This difference in response confirms Blind SQLi vulnerability. Further exploitation would involve crafting queries character-by-character (e.g., `' AND substring(database(),1,1)='d`).

4. **Blind SQL Injection (Burp Suite Intruder - DVWA Low Security):**
   a. Ensure Burp Proxy is capturing traffic.
   b. On the DVWA "SQL Injection (Blind)" page, submit a request with User ID `1`.
   c. In Burp Proxy > HTTP History tab, find the GET request (e.g., `/vulnerabilities/sqli_blind/?id=1&Submit=Submit#`).
   d. Right-click the request and "Send to Intruder".
   e. Go to the Intruder > Positions tab. Click "Clear §". Highlight the value `1` in the `id=1` parameter and click "Add §". Attack type should be "Sniper".
   f. Go to the Intruder > Payloads tab. Payload type: Simple list.
   g. Add potential Blind SQLi payloads (or load a list). Examples shown in screenshots:
   `' AND 1=1 --`
   `' AND 1=2 --`

`' OR 'a'='a`
`' having 1=1 --`
`' ORDER BY 1--`
`' ORDER BY 2--` (and potentially higher numbers)
h. Click "Start attack". A new window opens.
i. Observe the results table. Look for differences in "Length" (response size) or "Status" code corresponding to different payloads. Payloads causing TRUE conditions (e.g., `' AND 1=1 --`) should yield different responses than those causing FALSE (`' AND 1=2 --`) or errors (`' ORDER BY 3 --` if only 2 columns). (Screenshots show varying lengths like 4704, 4734, 4770, 4771).

Observations

- DVWA interface clearly shows input fields and output areas for SQLi testing.
- Low security setting performs minimal input sanitization, allowing basic SQLi payloads to work directly.
- Standard SQLi directly reveals database information in the web page output.
- Blind SQLi requires inferring information from application's binary responses (exists/missing).
- Burp Proxy successfully intercepts HTTP requests.
- Burp Intruder automates sending multiple payloads to the target parameter.
- Burp Intruder results table highlights differences in response length, indicating successful injection points for Blind SQLi.

Results/Calculations

- **Standard SQLi:** Successfully retrieved database version, name, user, hostname, and data directory. Confirmed existence of multiple users via tautology.
- **Blind SQLi:** Confirmed vulnerability manually via differing responses to TRUE/FALSE conditions. Confirmed vulnerability via Burp Intruder by observing varying response lengths for different payloads.

Discussion/Analysis

- SQL injection works by breaking out of the intended data context and introducing SQL commands, exploiting improperly constructed queries (often involving string concatenation).
- Union-based injection leverages the `UNION` operator to append results from a crafted query to the legitimate query's output. Requires knowing the number of columns.
- Blind SQLi is necessary when error messages or direct output are suppressed. It's slower but equally dangerous. Boolean-based blind SQLi uses TRUE/FALSE

conditions. Time-based blind SQLi uses database delay functions (`sleep()` , `benchmark()`).

- Burp Suite significantly speeds up testing, especially for Blind SQLi, by automating payload injection and highlighting response variations.
- **Prevention:**
  - **Prepared Statements (Parameterized Queries):** Best defense. Treats user input strictly as data, not executable code.
  - **Input Validation/Sanitization:** Whitelisting allowed characters/formats, escaping special characters (like `'`). Less reliable than prepared statements if not done perfectly.
  - **Least Privilege:** Connect to DB with users having minimal necessary permissions.
  - **Web Application Firewalls (WAFs):** Can detect/block common SQLi patterns, but can sometimes be bypassed.

Conclusion

- Successfully understood and demonstrated both standard (Union-based) and Blind SQL Injection vulnerabilities using DVWA.
- Performed attacks manually and leveraged Burp Suite Intruder to automate testing for Blind SQLi.
- Recognized the critical impact of SQLi and the importance of secure coding practices (especially prepared statements) for prevention.

Experiment 7: Introduction to OWASP and Implementation of XSS

Objective

- To understand Cross-Site Scripting (XSS) vulnerabilities.
- To understand the concept and purpose of OWASP and DVWA.
- To explore and exploit Reflected, Stored, and DOM-based XSS vulnerabilities in web applications using DVWA at different security levels.
- To understand basic XSS prevention techniques.

Background Theory

- **OWASP (Open Web Application Security Project):** A non-profit foundation focused on improving software security. Provides resources like documentation, tools, forums, and the famous "OWASP Top 10" list of critical web application security risks. Mission: make software security visible.
- **DVWA (Damn Vulnerable Web Application):** A tool for learning/teaching web application security, providing a legal platform to practice identifying and exploiting vulnerabilities like XSS.
- **Cross-Site Scripting (XSS):** A type of injection vulnerability where malicious scripts (usually client-side JavaScript) are injected into otherwise trusted websites. These scripts execute in the victim's browser when they visit the compromised page.
- **Types of XSS:**
  - **Stored (Persistent) XSS:** Malicious script is permanently stored on the target server (e.g., in a database via comment fields, forum posts, user profiles). Executes whenever a user visits the page containing the stored script. Most damaging type.
  - **Reflected (Non-Persistent) XSS:** Malicious script is embedded in a URL or request data. It is reflected off the web server to the victim's browser (e.g., in search results, error messages). Requires the victim to click a malicious link or submit a crafted form.
  - **DOM-based XSS:** Vulnerability exists in the client-side code (JavaScript) rather than server-side. Malicious script manipulates the Document Object Model (DOM) environment in the victim's browser, causing the client-side code to execute the script. Often involves unsafe handling of URL fragments (`#`) or data from `document.location`.
- **Impact of XSS:** Session hijacking (stealing cookies), performing actions on behalf of the user, phishing for credentials, keylogging, defacing websites, redirecting users to malicious sites.

Materials Used

- Web Browser
- DVWA Instance
- Burp Suite (Optional, but used in procedure)
- Referenced Websites: dvwa.co.uk, ensurtec.com, YouTube

Procedure

1. **Setup:**
   a. Access DVWA, log in (admin/password).

b. Set DVWA Security level to "Low".
c. (Optional) Configure browser to use Burp Suite Proxy.

2. **Reflected XSS (Low Security):**
   a. Navigate to "XSS (Reflected)".
   b. Input basic text `test` in the "What's your name?" field. Observe output `Hello test`.
   c. Input payload: `<script>alert('XSS')</script>`. Observe a JavaScript alert box appearing.
   d. Input HTML injection: `<b><h1>test</h1></b>`. Observe "test" displayed as a bold heading.
   e. *(Using Burp)* Intercept the request with `name=test`. Send to Repeater. Change `test` to `<script>alert('XSS_ATTACK')</script>`. Send the request. Observe the script in the response and the alert in the rendered view.
3. **Stored XSS (Low Security):**
   a. Navigate to "XSS (Stored)".
   b. Submit a normal entry: Name `Test User`, Message `Hello world`. Sign Guestbook. Observe the entry is displayed.
   c. Submit a malicious entry: Name `Attacker`, Message `<script>alert('Stored XSS')</script>`. Sign Guestbook.
   d. Observe the JavaScript alert box appearing immediately and on subsequent page loads/visits by any user.
   e. *(Using Burp)* Intercept the POST request to `/vulnerabilities/xss_s/`. Modify the `txtName` or `mtxMessage` parameter to include the XSS payload (`<script>alert('XSS_ATTACK')</script>`). Forward the request. Observe the stored payload executing.
4. **DOM-based XSS (Low Security):**
   a. Navigate to "XSS (DOM)".
   b. Select a language from the dropdown (e.g., English). Observe the URL changes to include `#English`.
   c. Manually change the URL in the browser's address bar to include a script in the fragment: `http://<dvwa_ip>/vulnerabilities/xss_d/?default=English#<script>alert('DOM XSS')</script>`.
   d. Hit Enter or trigger the relevant JavaScript. Observe a JavaScript alert box appearing.
5. **Exploring Medium Security:**
   a. Set DVWA Security to "Medium".
   b. *Reflected:* Re-try `<script>alert('XSS')</script>`. It likely fails. Analyze page source - medium often uses `str_replace('<script>', '', $name)`. Try payloads that bypass this, e.g., `<ScRiPt>alert('XSS')</ScRiPt>` (case change) or using different tags/events like `<img src=x onerror=alert('XSS')>`. (User notes input field restricted

to 10 chars, potentially limiting payloads).
c. *Stored:* Re-try `<script>alert('XSS')</script>`. Likely fails due to similar filtering on name/message. Try alternative payloads like `<img src=x onerror=alert(1)>`. (User notes name field 10 chars, message 100 chars).

6. **Exploring High Security:**
a. Set DVWA Security to "High".
b. *Reflected/Stored:* Analyze source - high often uses robust filtering (e.g., regex `preg_replace` removing tags) or output encoding (`htmlspecialchars`). Basic payloads and simple bypasses likely fail. (User notes character restriction 10, suggests trying `<img>` tag as `<script>` doesn't work). Exploitation might require more advanced techniques or might not be possible.
7. **(Related) CSRF:** The user also noted successfully changing the admin password from 'password' to 'pass' using the Cross-Site Request Forgery (CSRF) page in DVWA.

Observations

- Alert boxes (`alert( ... )`) successfully appeared when injecting `<script>` tags in Low security for Reflected, Stored, and DOM XSS.
- HTML tags like `<b>` and `<h1>` were rendered when injected in Low security Reflected XSS.
- Medium security appeared to filter the exact string `<script>`, requiring alternative payloads or case variations. Character limits were observed.
- High security appeared to have stronger filtering, making basic XSS difficult. Character limits persisted.
- Burp Suite allowed easy modification of HTTP requests to inject payloads for both Reflected and Stored XSS.

Results/Calculations

- Successful execution of JavaScript via XSS in Low security mode for all three types (Reflected, Stored, DOM).
- Demonstrated basic payload injection (`<script>alert( ... )`) and HTML injection.
- Analysis suggested increasing difficulty and need for payload variation/bypasses at Medium and High security levels due to filtering/encoding.

Discussion/Analysis

- **Reflected XSS:** Exploits trust in the server reflecting user input. Requires social engineering (getting victim to click link).

- **Stored XSS:** Most dangerous as script persists on server, affecting multiple users without interaction beyond visiting the page.
- **DOM XSS:** Vulnerability lies purely in client-side script's handling of data (often from URL). Server may be unaware.
- **Security Levels:** Demonstrate defense-in-depth. Low = no defense. Medium = basic blacklist filtering (often bypassable). High = stronger filtering/encoding (harder to bypass).
- **Prevention Techniques:**
  - **Input Validation/Sanitization:** Define allowed characters/formats (whitelist preferred over blacklist). Escape or remove dangerous characters server-side.
  - **Output Encoding:** Encode data before rendering it on the page, so browsers treat it as literal text, not executable code (e.g., `htmlspecialchars()` in PHP, context-aware encoding libraries). Crucial defense.
  - **Content Security Policy (CSP):** HTTP header defining allowed sources for content (scripts, styles, images). Can prevent execution of inline scripts or scripts from untrusted domains.
  - **HTTP-Only Cookies:** Prevents client-side JavaScript from accessing sensitive session cookies, mitigating session hijacking impact.

Conclusion

- Learned about OWASP and the concept of XSS vulnerabilities (Stored, Reflected, DOM-based).
- Successfully exploited these vulnerabilities in DVWA's Low security setting using basic JavaScript payloads.
- Observed how different security levels implement defenses (filtering, encoding) and increase the difficulty of exploitation.
- Understood the importance of context-aware output encoding and other measures (CSP, input validation) for preventing XSS attacks.

Experiment 8: Report Writing on Legal Issues and Ethics

Objective

- CO5: Interpret legal and ethical issues in security.
- To understand the complexities of legal and ethical issues in information security through case studies.
- To assess ethical dilemmas arising in professional practice.

- To apply ethical decision-making models and standards like the ACM Code of Ethics to real-world scenarios.

Expected Outcome

- Understand complexities of legal/ethical issues in InfoSec.
- Assess ethical dilemmas in coding/software development contexts.
- Apply ethical decision-making models.
- Align decisions with legal and ethical standards (e.g., ACM Code).

Background Theory

- **Ethics in Information Security:** Guiding principles for managing user data, designing software, and ensuring system security. Focuses on transparency, privacy, integrity, fairness, accountability, and minimizing harm.
- **Legal Compliance:** Adherence to applicable laws and regulations governing technology use, including:
  - Data Protection Laws (e.g., GDPR, CCPA)
  - Privacy Regulations
  - Intellectual Property Rights (Copyright, Patents)
  - Security Standards (e.g., PCI-DSS, HIPAA)
- **ACM Code of Ethics:** A framework defining ethical obligations for computing professionals. Key principles include: Contribute to society, Avoid harm, Be honest and trustworthy, Be fair and take action not to discriminate, Respect the work required to produce new ideas, Respect privacy, Honor confidentiality, Honor property rights (copyrights, patents).
- **Ethical Decision-Making Model (General Steps):**
  1. **Identify the Ethical Dilemma:** What is the core moral issue?
  2. **Identify Stakeholders:** Who is affected by the decision and how?
  3. **Consider Possible Actions:** What are the options? Analyze potential consequences (positive/negative) for each stakeholder.
  4. **Make a Decision:** Choose the option that best minimizes harm and aligns with ethical principles/codes.
  5. **Justify and Implement:** Document reasoning and act.
- **Legal Implications in Technology:** Unethical practices (violating privacy, IP, data protection laws) can lead to legal action, fines, reputational damage, and loss of trust.

Materials Used

- Case Study descriptions (provided in lab report: Case 1 - Sam Smith, Case 2 - Joseph, Case 3 - Sally Smith)
- ACM Code of Ethics documentation (acm.org/code-of-ethics)
- Referenced articles/websites on ethics (ERIC, PMI, pressbooks).

Procedure: Case Study Analysis

Case 1: Sam Smith (Disruptive Member vs. Org Integrity)

1. **Hypothetical Project:** Ethics Violation Detection System (EVDS) to monitor communications for unethical behavior (misleading statements, false accusations, disruption).
2. **Scenario:** Long-time member Sam Smith disrupts meetings, sends accusatory emails, files rejected ethics complaints, violates a settlement agreement by trying to rejoin misleadingly. Causes operational/financial impact.
3. **Dilemma:** Sam's right to express concerns vs. Organization's need to maintain integrity, fairness, and prevent harassment/damage.
4. **Stakeholders:** Sam, PMI Fredonia Chapter, PMI GHQ, Board Members, Local Authorities, Other Members.
5. **Options Analysis:**
   - *Allow Rejoin:* (+) Sam gets 2nd chance. (-) Risk of repeat behavior, loss of credibility, more disruption.
   - *Permanent Ban:* (+) Protects org, prevents future issues. (-) Sam fully excluded.
   - *Implement EVDS:* (Neutral) Ensures fair monitoring. (+) Strengthens ethical compliance, protects board, transparent environment. (-) Potential privacy concerns if not implemented carefully.
6. **Decision:** Implement EVDS. Best balances fairness, protection, and organizational integrity.
7. **ACM Alignment:** Aligns with 1.1 (Well-being), 1.2 (Avoid Harm), 1.3 (Honest/Trustworthy), 2.5 (Evaluations), 3.5 (Privacy - if done right).
8. **Scenario Fix:** Clear Guidelines, EVDS, Independent Review, Tiered Restrictions, Binding Mediation.

Case 2: Joseph (Abusive Communication vs. Professional Conduct)

1. **Hypothetical Project:** Automated Communication Monitoring System (ACMS) to detect harassment, excessive messaging, unprofessional behavior.
2. **Scenario:** Member Joseph denied use of PMI domain name, responds with excessive/rude/threatening texts (dozens per day) to PMI staff and Chapter President. Violates respect (Code 3.3.3).

3. **Dilemma:** Joseph's right to express dissatisfaction vs. PMI staff's right to a respectful, non-abusive work environment.
4. **Stakeholders:** Joseph, PMI Staff, Chapter President, PMI GHQ, Other Members.
5. **Options Analysis:**
    - *Allow Messaging:* (+) No restriction on Joseph. (-) Staff harassment continues, admin burden, unethical environment.
    - *Permanent Ban:* (+) Protects staff, stops harassment. (-) Joseph loses PMI access.
    - *Implement ACMS:* (Neutral) Ensures fair monitoring. (+) Flags abusive messages automatically, automates complaint tracking, prevents future misconduct.
6. **Decision:** Implement ACMS. Maintains professional standards while detecting/handling unethical behavior automatically.
7. **ACM Alignment:** Aligns with 1.1 (Well-being), 1.2 (Avoid Harm), 1.3 (Honest/Trustworthy - encourages respectful correspondence), 1.4 (Fairness), 3.1 (Public Good - professional environment).
8. **Scenario Fix:** Clear Policies (volume/tone), ACMS, Escalation Mechanism, Warnings, Consequences (temp/perm bans).

Case 3: Sally Smith (Unintentional Plagiarism vs. IP Rights)

1. **Hypothetical Project:** Plagiarism Detection and Citation Validator System (PDCVS) to scan articles for plagiarism and validate citations.
2. **Scenario:** Author Sally Smith copies material without attribution in an article. Sam Jones reports it. Sally admits unintentional error due to miscommunication with publisher. ERC issues private reprimand for violating property rights (Code 3.3.4).
3. **Dilemma:** Respecting Intellectual Property vs. Handling unintentional errors fairly without overly harsh penalties.
4. **Stakeholders:** Sally Smith, Sam Jones, Original Author, Website Publisher, PMI/ERC, Readers/Professionals.
5. **Options Analysis:**
    - *Ignore:* (+) No immediate consequence for Sally. (-) No credit for original author, website loses credibility, misinformation spreads.
    - *Formal Sanction:* (+) Shows ethical enforcement. (-) Reputation damage for Sally (potentially harsh for unintentional error). (+) Gets recognition for original author.
    - *Implement PDCVS:* (Neutral) Future prevention. (+) Ensures proper credit, reduces future cases, improves trust in content.
6. **Decision:** Implement PDCVS. Prevents future plagiarism, ensures attribution, allows handling errors fairly.

7. **ACM Alignment:** Aligns with 1.5 (Respect Work), 1.6 (Honor Property Rights), 1.7 (Give Proper Credit), 3.1 (Public Good - reliable publications).
8. **Scenario Fix:** Integrate PDCVS, Educate Authors, Pre-Publication Checks, Correction Process, Proportional Consequences (distinguish accidental/intentional).

Observations

- Each case study presents a realistic ethical challenge in a professional/computing context.
- Applying a structured decision-making model helps clarify dilemmas, stakeholders, options, and impacts.
- The ACM Code of Ethics provides relevant principles for guiding decisions in these scenarios.
- Technology can potentially aid enforcement (EVDS, ACMS, PDCVS) but requires careful implementation regarding fairness and privacy.

Results/Calculations

- Formulated ethical decisions for each case study based on analysis.
- Justified decisions by aligning them with specific principles from the ACM Code of Ethics.
- Proposed concrete steps to "fix" the scenarios to ensure future legal and ethical compliance.

Discussion/Analysis

- Ethical decision-making in information security often involves balancing competing values (e.g., free speech vs. preventing harassment, intellectual property vs. handling errors).
- Professional codes like the ACM Code provide essential guidance but require interpretation and application to specific contexts.
- Transparency, fairness, accountability, and minimizing harm are recurring themes in ethical solutions.
- Legal compliance often overlaps with ethical considerations (e.g., respecting privacy and data protection laws).
- Automated systems for monitoring/detection must be implemented carefully to avoid bias and protect individual rights.

Conclusion

- Learned the importance of ethical decision-making, accountability, and maintaining professional integrity in information security and project management contexts.
- Successfully analyzed three distinct case studies involving ethical dilemmas (disruptive behavior, abusive communication, plagiarism) using an ethical decision-making framework and the ACM Code of Ethics.
- Understood how to identify stakeholders, evaluate options, and align decisions with professional standards.

Experiment 9: Illustrate and Compare Network Security Mechanisms (Wireshark & Network Miner)

Objective

- To analyze network traffic from a real-life case study using network security/forensic tools: Wireshark and Network Miner.
- To compare the capabilities and use cases of Wireshark and Network Miner in investigating network-based incidents.
- To extract specific artifacts and information from a packet capture (PCAP) file.

Expected Outcome

- CO4: Illustrate and Compare network security mechanisms. Student will be able to use Wireshark and Network Miner to analyze network traffic and compare their functionalities.

Background Theory

- **Wireshark:** A powerful and widely used open-source network protocol analyzer (packet sniffer).
  - **Functionality:** Captures live traffic or reads saved PCAP files. Dissects packets layer-by-layer, showing detailed protocol information (Ethernet, IP, TCP/UDP, Application layers). Supports hundreds of protocols. Offers powerful display and capture filters. Can reconstruct TCP streams.
  - **Use Cases:** Network troubleshooting, software/protocol development, security analysis (detecting anomalies, intrusions), network forensics, learning network protocols.
- **Network Miner:** A passive network forensic analysis tool (NFAT). Focuses on extracting application-level data and artifacts from PCAP files or live traffic without

deep packet-by-packet inspection required by the user.
    - **Functionality:** Passive sniffing (doesn't inject packets). Automatically extracts artifacts like files, images, emails, credentials, SSL certificates, DNS queries, sessions, host information (OS, hostname). Presents data in a host-centric view. User-friendly GUI.
    - **Use Cases:** Rapid incident response (quick evidence collection), extracting compromised data, analyzing large PCAP files efficiently, complementing Wireshark's detailed analysis.
- **Packet Sniffing:** The process of intercepting and logging traffic passing over a digital network. Can be used legitimately for network management/monitoring or maliciously for stealing information (passwords, sensitive data).
- **PCAP (Packet Capture):** Standard file format for storing captured network traffic.

Materials Used

- Wireshark software
- Network Miner software
- PCAP file: `evidence01.pcap` (from forensicscontest.com, "Ann's Bad Aim" puzzle)
- Computer System
- Reference: forensicscontest.com puzzle solution page

Procedure: Analyzing `evidence01.pcap`

Using Wireshark:

1. Open `evidence01.pcap` in Wireshark.
2. Identify key IP addresses involved in conversations. (User identified external IP `64.12.24.59`).
3. Apply a display filter to focus on traffic involving this IP: `ip.addr == 64.12.24.59`.
4. Examine the protocols involved (likely TCP/SSL on port 443 for IM).
5. Right-click on relevant packets (likely related to an IM conversation) and select "Follow > TCP Stream" (or SSL Stream if TLS decryption keys were available, which they usually aren't).
6. Analyze the reconstructed stream content:
    - Identify usernames/participants (Found `Sec558user1`).
    - Read the conversation text to find the first comment: "Here's the secret recipe... I just downloaded it from the file server. Just copy to a thumb drive and you're good to go >:-)".
    - Identify the filename mentioned: `recipe.docx`.

7. Locate the stream corresponding to the file transfer. Since it's likely encrypted (port 443), look for clues *around* the transfer.
8. Find the packet containing the start of the file data. Examine the hex view for the "magic number" (first few bytes) of the file type. Found `50 4b 03 04` (hex), which corresponds to the magic number for ZIP archives (and thus DOCX files).

Using Network Miner:

1. Open `evidence01.pcap` in Network Miner. (It processes the file upon opening).
2. Navigate through the tabs:
   - **Hosts:** View identified hosts based on traffic.
   - **Files:** Check for automatically extracted files. Locate `recipe.docx`.
     - Select `recipe.docx`. View its details (MD5 hash, source/destination IPs, timestamps). Found MD5: `8350582774e1d4dbe1d61d64c89e0ea1`.
     - Right-click and "Open file" or "Open folder" to access the extracted file.
   - **Messages/Emails:** Check if any emails or messages were extracted (might extract IM messages if protocol is supported and unencrypted, less likely here due to SSL).
   - **Credentials:** Check for any captured usernames/passwords (e.g., from HTTP Basic Auth, FTP, etc. - unlikely in this specific encrypted IM scenario).
   - **Images:** Check for extracted images.
   - **Sessions:** View reconstructed sessions.
3. Open the extracted `recipe.docx` file using appropriate software (e.g., Microsoft Word, LibreOffice Writer).
4. Read the content: "Recipe for Disaster..." including ingredients (4 cups sugar, 2 cups water) and instructions.

Observations

- Wireshark displayed thousands of packets, requiring filters to isolate relevant traffic.
- Following TCP streams in Wireshark revealed the IM conversation text and username.
- Wireshark's hex view allowed identification of the file's magic number (`504B0304`).
- Network Miner automatically parsed the PCAP and presented extracted artifacts in organized tabs.
- Network Miner successfully extracted `recipe.docx` directly from the encrypted stream (likely by reassembling TCP segments before TLS layer analysis failed, or by identifying common file transfer patterns).
- Network Miner provided the MD5 hash of the extracted file.

- The content of the extracted `recipe.docx` matched the expected "secret recipe".

Results/Calculations

- **Answers to Puzzle Questions:**
  1. Ann's IM buddy: `Sec558user1` (from Wireshark stream)
  2. First comment: "Here's the secret recipe... I just downloaded it from the file server. Just copy to a thumb drive and you're good to go >:-)" (from Wireshark stream)
  3. File name: `recipe.docx` (from Wireshark stream & Network Miner Files tab)
  4. Magic number (first 4 bytes): `0x504B0304` (from Wireshark hex view)
  5. MD5sum of file: `8350582774e1d4dbe1d61d64c89e0ea1` (from Network Miner file details)
  6. Secret recipe: "Recipe for Disaster: 1 serving, Ingredients: 4 cups sugar, 2 cups water..." (from extracted file content via Network Miner)
- Hashing (MD5) performed automatically by Network Miner.

Discussion/Analysis

- **Wireshark Strengths:** Deep packet inspection, detailed protocol analysis, identifying specific bytes/patterns (magic numbers), troubleshooting complex network issues. Essential for understanding *how* protocols work.
- **Network Miner Strengths:** Automated artifact extraction (files, creds, images), host-centric view, ease of use for quick evidence gathering, efficient for large PCAPs when focus is on extracted data rather than protocol details.
- **Comparison:** Wireshark is for low-level, detailed analysis. Network Miner is for higher-level, artifact-focused analysis. They are complementary tools in network forensics. Wireshark might be needed to find clues Network Miner misses, while Network Miner rapidly extracts readily available artifacts.
- **Challenges in Network Forensics:** High volume of data, encrypted traffic (major hurdle), real-time detection needs, sophisticated evasion techniques, attribution difficulties, resource constraints, false positives from monitoring tools.
- **Other Monitoring Tools:** Snort (IDS/IPS), Nmap (Network Scanner), Tcpdump (Command-line sniffer), SolarWinds NPM (Commercial), Nagios (Monitoring System).
- **Packet Sniffing:** Intercepting network data. Legitimate uses: troubleshooting, performance analysis. Malicious use: stealing credentials, sensitive info. Tools: Wireshark, Tcpdump.

Conclusion

- Successfully demonstrated the use of Wireshark and Network Miner to analyze a network capture file and solve a forensic challenge.
- Showcased Wireshark's strength in detailed packet/stream analysis for finding specific text and byte patterns (magic numbers).
- Showcased Network Miner's strength in automatically extracting artifacts like files and providing metadata (MD5 hash).
- Understood that combining multiple tools enhances the efficiency and accuracy of network forensic investigations.

Experiment 10: Digital Forensics Investigation using EnCase Tool

Objective

- To understand the fundamentals of digital forensics and its importance in cybersecurity.
- To learn how to use the EnCase forensic tool for acquiring (conceptually) and analyzing digital evidence from a forensic image.
- To simulate basic digital forensic investigation tasks using EnCase (browsing files, searching, viewing artifacts).
- To interpret results and understand the importance of evidence integrity and ethical guidelines.

Background Theory

- **Digital Forensics:** The process of preservation, identification, extraction, documentation, and interpretation of computer/digital evidence. Used in investigations involving cybercrime, data breaches, policy violations, and litigation.
- **EnCase:** A widely used commercial digital forensics platform developed by OpenText (formerly Guidance Software). Provides tools for forensic imaging (acquisition), analysis of storage media, file recovery, searching, reporting, etc.
- **Core Concepts:**
  - **Evidence Acquisition (Forensic Imaging):** Creating a bit-for-bit, verified copy (image) of the original storage media (e.g., hard drive, USB drive). Done using write-blockers to prevent altering the original evidence. Common formats: EnCase E01, RAW (dd).
  - **Hashing (MD5, SHA-1):** Cryptographic hashes are calculated for the original media (if possible) and the forensic image. They are compared to verify the

image is an exact, unaltered copy. Hashing is also used on individual files for integrity checks and identification (known good/bad lists).

- **Chain of Custody:** A documented record tracing the seizure, custody, control, transfer, analysis, and disposition of evidence. Ensures evidence integrity and admissibility in legal proceedings.
- **Artifact Analysis:** Identifying and examining data relevant to an investigation, including existing files, deleted files, file metadata (timestamps), registry entries, system logs, browser history, email, link files, etc.
- **File Signature Analysis:** Comparing the internal "magic number" (first few bytes identifying file type) with the file extension to detect disguised or corrupted files.
- **Keyword Searching:** Searching the entire evidence image (including unallocated space, slack space) for specific terms relevant to the case.
- **Reporting:** Documenting the entire forensic process, findings, and conclusions in a clear, concise, and legally acceptable format.

Materials Used

- EnCase Forensic Software (Version specific details not provided, but likely v7 or later)
- Forensic Image File(s) (e.g., `.E01`, `.dd`, or mounted drives within EnCase)
- Computer System capable of running EnCase
- Referenced Websites/Videos: opentext.com, forums.opentext.com, YouTube tutorials on EnCase features.

Procedure (Using EnCase - based on screenshots)

1. **Case Setup:** Launch EnCase. Create a New Case or Open an existing one. Provide case details (name, examiner name, etc.).
2. **Add Evidence:** Add the forensic image file(s) to the case via the "Add Evidence" function. EnCase automatically verifies the hash of the image upon adding (if available in the image format like E01).
3. **Evidence Processing (Optional but common):** Run predefined processing tasks like file signature analysis, hash analysis (comparing against known libraries), index creation for searching, metadata extraction, registry parsing etc.
4. **Explore Evidence Structure:** Use the Tree Pane (left side) to navigate the directory structure within the forensic image (e.g., drives, partitions, folders like `Company's USB`, `Secret_project_design`).
5. **Examine Files:**
   - Select folders/files in the Tree Pane or Table Pane (top right).

- View file listings in the Table Pane, showing metadata (name, size, dates, hash, etc.).
- Select individual files to view their content in the View Pane (bottom). Use different tabs/viewers: Text, Hex, Picture, Transcript, Doc View (renders documents). (Screenshots show viewing `.docx`, `.ppt`, `.pdf`).

6. **Keyword Searching:**
   - Go to the "Search" or "Keywords" section.
   - Define keywords relevant to the investigation (e.g., "secret", "password", "meeting", project names, usernames). (Screenshot shows predefined keywords like "My friends", "Password", "Unallocated Clusters").
   - Run the search across the evidence.
   - Review search hits, examining the context where keywords were found. (Screenshot shows search results list).
7. **File Signature Analysis:**
   - Navigate to the "File Types" view, often under a "View" or "Tools" menu.
   - Examine the "File Signature" column in the Table Pane or a dedicated view. EnCase compares the actual file header bytes with known signatures for the file's extension.
   - Identify files with mismatched signatures (e.g., a file with `.jpg` extension having a `.exe` or `.zip` header), indicating potential obfuscation or corruption. (Screenshot shows the File Types view with Header/Footer definitions and signature mismatches potentially flagged).
8. **Filtering and Sorting:** Utilize filters (e.g., show only picture files, files created in a specific date range) and sort columns in the Table Pane to manage large amounts of data and focus analysis.
9. **Bookmarking:** Right-click on relevant files, data, or search hits and select "Bookmark" to tag items of interest for later review or inclusion in the report.
10. **Reporting:** Use the "Report" function to generate a summary of findings, including case details, evidence information, analysis performed, and bookmarked items.

Observations

- EnCase provides a multi-pane interface (Tree, Table, View, Filters) for navigating and analyzing evidence.
- Evidence is added non-destructively (read-only access to image). Hash verification ensures integrity.
- Various viewers allow examining file content in different formats (text, hex, rendered documents, pictures).

- Keyword searching is a powerful tool for finding relevant information quickly across large datasets.
- File signature analysis helps identify potentially suspicious files regardless of their extension.
- The tool visually flags or allows filtering for items like files with bad signatures, known hash values, or keyword hits.

Results/Calculations

- Identification of relevant user files, documents, pictures within the forensic image.
- Discovery of potentially sensitive information through keyword searching (e.g., files related to `secret_project_design`).
- Identification of files with mismatched signatures (potential hiding of data).
- Hash values (MD5, SHA1) are calculated and used by EnCase for integrity verification and comparison against known hash sets.

Discussion/Analysis

- **Evidence Integrity:** EnCase maintains integrity primarily through initial hash verification of the forensic image and by accessing the image in a read-only manner. Hashing individual files allows tracking changes and comparing against known databases.
- **File Signature Analysis:** Crucial because file extensions can be easily changed by users to hide files. Comparing the internal signature (magic number) provides a more reliable way to determine the true file type and identify suspicious files (e.g., malware disguised as an image).
- **Search Tools:** Keyword searching allows targeted searching for specific terms. Filters (by file type, date, path, hash set, etc.) help narrow down the scope of investigation from potentially millions of files to a manageable set. Metadata/Path views help understand context.
- **Ethical Guidelines:** Extremely important in digital forensics. Investigators handle sensitive, private, and potentially incriminating data. Adherence to ethics is crucial for:
  - Protecting individual rights (privacy).
  - Ensuring evidence admissibility in court (improper handling can lead to dismissal).
  - Maintaining professional integrity and public trust.
  - Avoiding legal consequences and reputational damage from actions like fabricating/modifying evidence or accessing data without cause. Following protocols upholds the investigation's credibility and legality.

Conclusion

- Learned the fundamental principles of digital forensics, including evidence preservation, integrity verification (hashing), and artifact analysis.
- Gained familiarity with the EnCase forensic tool's interface and basic functionalities for adding evidence, navigating file systems within an image, performing keyword searches, and conducting file signature analysis.
- Understood the importance of maintaining evidence integrity and adhering to strict ethical guidelines throughout a digital forensic investigation.

Oral Exam Questions & Answers

1. **What is the core difference between a substitution cipher and a transposition cipher?**
   - **Answer:** Substitution ciphers replace letters/symbols with others, while transposition ciphers rearrange the order of the original letters/symbols.
2. **Explain the primary vulnerability of the Caesar cipher.**
   - **Answer:** It's vulnerable to brute-force attack because there are only 25 possible meaningful shifts to try for the English alphabet.
3. **How does a keyed columnar transposition cipher determine the order of columns?**
   - **Answer:** The columns are read out based on the alphabetical order of the letters in the keyword.
4. **In the Caesar cipher C++ code, what is the purpose of the line `newPos = (26 + ((pos + shift) % 26)) % 26;`?**
   - **Answer:** It calculates the new character position after applying the shift, ensuring correct wrap-around within the 0-25 range of the alphabet using modular arithmetic, robustly handling positive and negative shifts.
5. **What does RSA stand for, and what mathematical problem does its security rely on?**
   - **Answer:** RSA: Rivest-Shamir-Adleman. Its security relies on the computational difficulty of factoring large prime numbers.
6. **List the components of an RSA public key and an RSA private key.**
   - **Answer:** Public Key: `(n, e)` (modulus `n`, public exponent `e`). Private Key: `(n, d)` (modulus `n`, private exponent `d`).
7. **What are the five main steps in the RSA key generation process?**
   - **Answer:** 1. Prime Selection (p, q), 2. Modulus Computation (n=p*q), 3. Euler Totient Calculation (φ(n)=(p-1)(q-1)), 4. Public Exponent Selection (e), 5. Private

Exponent Computation (d).

8. **What OpenSSL command is used to generate an RSA private key?**
   - **Answer:** `openssl genpkey -algorithm RSA -out private_key.pem`
9. **What OpenSSL command is used to extract the public key from a private key file?**
   - **Answer:** `openssl rsa -pubout -in private_key.pem -out public_key.pem`
10. **In the RSA post-lab calculation (p=7, q=11, e=13), what was the calculated value of `d`?**
    - **Answer:** `d = 37`.
11. **What is a digital signature, and what security properties does it provide in the context of RSA/PGP?**
    - **Answer:** A digital signature is created by encrypting a hash of the message with the sender's private key. It provides Authentication (proves sender identity), Integrity (ensures message wasn't altered), and Non-repudiation (sender cannot deny sending).
12. **Explain two practical limitations of RSA.**
    - **Answer:** Two limitations: 1. Slow Computation compared to symmetric algorithms. 2. Large Key Size required for security (2048+ bits). (Others: Vulnerability to Factorization/Quantum, Side-channel attacks, No Forward Secrecy).
13. **What does CAPTCHA stand for?**
    - **Answer:** CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart.
14. **Describe two different types of CAPTCHA implemented in Experiment 3.**
    - **Answer:** Two types implemented: Text-based (typing distorted text), Mathematical (solving simple math), Distorted Text/Canvas (typing text rendered with distortion on canvas), Image Grid (selecting images matching a description). (Any two).
15. **What is the main security goal of using CAPTCHA on a website?**
    - **Answer:** To prevent automated bots from performing actions like spamming, scraping data, or attempting brute-force logins, by ensuring the user is human.
16. **In the Text CAPTCHA JavaScript, how was the generated random string stored for verification?**
    - **Answer:** It was stored in a data attribute of the HTML paragraph element displaying the CAPTCHA (`element.dataset.captcha`).
17. **In the Grid Image CAPTCHA JavaScript, how were images marked as 'selected' by the user?**
    - **Answer:** A click event listener was added to each image, which toggled a CSS class (`selected`) on the clicked image.

18. **What is a primary accessibility issue with many visual CAPTCHAs?**
    - **Answer:** They can be difficult or impossible for users with visual impairments to solve.
19. **What is reCAPTCHA, and how does it often differ from traditional CAPTCHA?**
    - **Answer:** reCAPTCHA (by Google) often uses behavioral analysis and risk scoring to differentiate humans from bots, sometimes being invisible, whereas traditional CAPTCHA relies on direct challenges (text, images, math).
20. **What is PGP, and what standard does it typically follow for email encryption?**
    - **Answer:** PGP: Pretty Good Privacy. It follows the OpenPGP standard (RFC 4880).
21. **Explain the concept of End-to-End Encryption (E2EE) as provided by Mailvelope/PGP.**
    - **Answer:** E2EE means the message is encrypted by the sender and can *only* be decrypted by the intended recipient(s)' private key(s). Intermediate servers (like email providers) cannot read the content.
22. **What role does the private key play in PGP when sending a signed email?**
    - **Answer:** The private key is used to create the digital signature (by encrypting a hash of the message).
23. **What role does the public key play in PGP when sending an encrypted email?**
    - **Answer:** The recipient's public key is used to encrypt the message so only they can decrypt it with their private key.
24. **What browser extension was used in Experiment 4 to implement PGP in webmail?**
    - **Answer:** Mailvelope.
25. **Why is key management important for PGP security?**
    - **Answer:** Security relies on keeping the private key secret (protected by password) and verifying the authenticity of contacts' public keys to prevent impersonation or MITM attacks.
26. **What is `iptables` used for in Linux?**
    - **Answer:** To configure the Linux kernel firewall (Netfilter), defining rules to filter network packets.
27. **Name the three main tables used in `iptables`.**
    - **Answer:** `filter`, `nat`, `mangle`.
28. **What are the three main built-in chains in the `filter` table of `iptables`?**
    - **Answer:** `INPUT`, `OUTPUT`, `FORWARD`.
29. **What is the difference between the `DROP` and `REJECT` targets in `iptables`?**
    - **Answer:** DROP silently discards the packet. REJECT discards the packet but sends an error notification (e.g., ICMP port unreachable) back to the sender.
30. **What `iptables` command parameter and state(s) are used to allow return traffic for established connections?**

- **Answer:** The `-m state --state ESTABLISHED,RELATED` parameters are used to match such traffic, typically with a target of `ACCEPT`.

31. **What target is used in the `nat` table's `POSTROUTING` chain to allow internal networks to access the internet using the firewall's IP address?**
    - **Answer:** `MASQUERADE`.
32. **What command is used to save the current `iptables` rules to a file?**
    - **Answer:** `sudo iptables-save > /path/to/rules/file`
33. **What is SQL Injection (SQLi)?**
    - **Answer:** A web security vulnerability where an attacker inserts malicious SQL code into input fields, which is then executed by the backend database.
34. **What is the difference between In-Band (Standard) SQLi and Blind SQLi?**
    - **Answer:** In-Band (Standard) SQLi: The attacker retrieves data directly through the application's response (e.g., Union-based). Blind SQLi: The attacker infers data by observing the application's behavior (e.g., true/false responses, time delays) as data isn't returned directly.
35. **What tool was used in Experiment 6 to automate the sending of multiple SQLi payloads? What specific feature was used?**
    - **Answer:** Burp Suite was used, specifically the Intruder feature.
36. **In the manual standard SQLi procedure, what payload was used to retrieve the database version?**
    - **Answer:** `1' union select null,@@version #` (or similar syntax depending on DB).
37. **What is the primary defense mechanism recommended against SQL Injection?**
    - **Answer:** Prepared Statements (Parameterized Queries).
38. **What does OWASP stand for?**
    - **Answer:** OWASP: Open Web Application Security Project.
39. **Explain the difference between Stored XSS and Reflected XSS.**
    - **Answer:** Stored XSS: Malicious script is saved on the server (e.g., in a database) and served to victims who visit the page. Reflected XSS: Malicious script is included in a request (e.g., URL parameter) and reflected back in the server's response to the victim's browser.
40. **What is DOM-based XSS?**
    - **Answer:** A type of XSS where the vulnerability exists in the client-side JavaScript code that manipulates the Document Object Model (DOM) unsafely, often using data from the URL fragment (`#`).
41. **What simple JavaScript function was commonly used in the XSS payloads to demonstrate successful injection?**
    - **Answer:** `alert('some message')` or `alert(1)`.

42. **Mention two common techniques used by web applications (like DVWA Medium/High levels) to prevent XSS.**
    - **Answer:** Two techniques: 1. Input Sanitization/Filtering (e.g., removing `<script>` tags, using regex - often done in Medium/High). 2. Output Encoding (e.g., using `htmlspecialchars` to treat input as text - often in High/Impossible). (Also Content Security Policy).
43. **What is the purpose of the ACM Code of Ethics?**
    - **Answer:** To provide a guiding framework defining the ethical obligations and standards of behavior for professionals in the computing field.
44. **According to the ethical decision-making model used in Experiment 8, what are the first two steps when facing an ethical dilemma?**
    - **Answer:** 1. Identify the Ethical Dilemma. 2. Identify Stakeholders.
45. **In the analysis of Case 3 (Sally Smith - Plagiarism), what ACM principle was primarily violated?**
    - **Answer:** 1.6 Honor property rights including copyrights and patents (implicitly violated by plagiarism) and 3.3.4 (Respect) We respect the property rights of others (explicitly mentioned in case description analysis).
46. **What is the main difference in focus between Wireshark and Network Miner when analyzing network traffic?**
    - **Answer:** Wireshark focuses on detailed, low-level packet analysis and protocol dissection. Network Miner focuses on higher-level, automated extraction of artifacts (files, images, credentials, metadata) from traffic.
47. **In Experiment 9's case study, what tool was used to find the MD5 hash of the `recipe.docx` file?**
    - **Answer:** Network Miner (in the Files tab details).
48. **What is a "magic number" in the context of file analysis (Experiment 9 & 10)?**
    - **Answer:** The first few bytes of a file that uniquely identify the file type (e.g., `50 4B 03 04` for ZIP/DOCX, `FF D8` for JPEG). Used in file signature analysis.
49. **What is the primary purpose of calculating and verifying hashes (MD5/SHA1) in digital forensics (Experiment 10)?**
    - **Answer:** To verify the integrity of digital evidence – ensuring that the forensic image is an exact copy of the original and that evidence files have not been altered during analysis.
50. **Why is file signature analysis important in EnCase (or digital forensics in general)?**
    - **Answer:** Because file extensions can be easily changed by users to disguise files. Signature analysis compares the internal magic number to the extension to detect mismatched/hidden files (e.g., malware disguised as a .jpg).