## Analyze and visualize IP address allocation and subnetting using a Python script with Scapy and Matplotlib.

Use in command

1. pip install scapy matplotlib numpy

file n ame = ip\_analysis

from scapy.all import sniff

from scapy.layers.inet import IP # Explicit import

from collections import Counter

import matplotlib.pyplot as plt

import ipaddress

# Counter to track IP occurrences

ip\_count = Counter()

# Function to capture packets

def capture\_packets(packet):

if packet.haslayer(IP): # Correcting IP reference

ip\_count[packet[IP].src] += 1

ip\_count[packet[IP].dst] += 1

# Capture 50 packets

sniff(prn=capture\_packets, count=50)

# Visualization of IP address allocation

ips = list(ip\_count.keys())

counts = list(ip\_count.values())

plt.figure(figsize=(10, 5))

plt.bar(ips, counts, color='blue')

plt.xticks(rotation=45, ha="right") # Improve readability

plt.xlabel("IP Addresses")

plt.ylabel("Number of Packets")

plt.title("IP Address Allocation Analysis")

plt.tight\_layout() # Adjust layout for better visibility

plt.show()

# Function to classify IP as public or private

def classify\_ip(ip):

ip\_obj = ipaddress.ip\_address(ip)

return "Private IP" if ip\_obj.is\_private else "Public IP"

# Print IP classifications

for ip in ip\_count.keys():

print(f"{ip}: {classify\_ip(ip)}")

1. **Develop a Python program to read and modify Linux network configuration files such as /etc/hosts, /etc/resolv.conf, and /etc/network/interfaces.**

**Filw name =** network\_config

**import os**

**import shutil**

**# Windows hosts file location**

**HOSTS\_FILE = r"C:\Windows\System32\drivers\etc\hosts"**

**def read\_hosts():**

**"""Reads and displays the contents of the hosts file."""**

**if not os.path.exists(HOSTS\_FILE):**

**print("[ERROR] Hosts file not found!")**

**return**

**try:**

**with open(HOSTS\_FILE, 'r') as file:**

**print("\n===== Contents of hosts file =====\n")**

**print(file.read())**

**except Exception as e:**

**print(f"[ERROR] Failed to read hosts file: {e}")**

**def modify\_hosts():**

**"""Backs up and modifies the hosts file."""**

**if not os.path.exists(HOSTS\_FILE):**

**print("[ERROR] Hosts file not found!")**

**return**

**try:**

**# Backup the hosts file before modifying**

**backup\_file = HOSTS\_FILE + ".bak"**

**shutil.copy(HOSTS\_FILE, backup\_file)**

**print(f"[INFO] Backup created: {backup\_file}")**

**# Append a new entry**

**with open(HOSTS\_FILE, "a") as file:**

**file.write("\n192.168.1.100 mycustomhost\n")**

**print("[SUCCESS] Hosts file updated.")**

**except Exception as e:**

**print(f"[ERROR] Failed to update hosts file: {e}")**

**def restart\_network():**

**"""Flushes the DNS cache to apply changes."""**

**try:**

**os.system("ipconfig /flushdns")**

**print("[SUCCESS] DNS cache flushed. Changes applied.")**

**except Exception as e:**

**print(f"[ERROR] Failed to flush DNS cache: {e}")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Step 1: Read and display the hosts file**

**read\_hosts()**

**# Step 2: Modify the hosts file**

**modify\_hosts()**

**# Step 3: Flush DNS cache to apply changes**

**restart\_network()**

1. **Write a Python script to capture and analyze active TCP/IP daemons running on a system using psutil and netstat.**

**import psutil**

**def list\_tcp\_connections():**

**"""Lists all active TCP connections."""**

**for conn in psutil.net\_connections(kind='tcp'):**

**print(f"PID: {conn.pid}, Local Address: {conn.laddr}, Remote Address: {conn.raddr}, Status: {conn.status}")**

**# Call the function**

**list\_tcp\_connections()**

1. **Create a simple Python-based network daemon that listens on a specified port and logs incoming connections.**

**import socket**

**def start\_daemon(host='0.0.0.0', port=5000):**

**server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**server\_socket.bind((host, port))**

**server\_socket.listen(5)**

**print(f"Listening on {host}:{port}")**

**while True:**

**client\_socket, client\_address = server\_socket.accept()**

**print(f"Connection received from {client\_address}")**

**client\_socket.sendall(b"Hello! You are connected.\n")**

**client\_socket.close()**

**start\_daemon()**

1. **Develop a Python script to scan and list open ports on a target machine using socket**

**and nmap libraries.**

**Intsall = pip install python-nmap**

**Not work means**

**python3 port\_scanner.py**

**or**

**netstat -tulnp**

**import socket**

**import nmap**

**# Function to scan ports using socket**

**def scan\_ports(target, ports):**

**print(f"Scanning {target} using socket...")**

**for port in ports:**

**try:**

**sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**sock.settimeout(1)**

**result = sock.connect\_ex((target, port))**

**if result == 0:**

**print(f"Port {port} is OPEN")**

**sock.close()**

**except Exception as e:**

**print(f"Error scanning port {port}: {e}")**

**# Function to scan ports using nmap**

**def nmap\_scan(target):**

**print(f"\nScanning {target} using Nmap...")**

**try:**

**scanner = nmap.PortScanner()**

**scanner.scan(target, '1-1024', '-sS')**

**for port in scanner[target]['tcp']:**

**state = scanner[target]['tcp'][port]['state']**

**print(f"Port {port}: {state.upper()}")**

**except Exception as e:**

**print(f"Error with Nmap scan: {e}")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**target\_ip = "192.168.1.1" # Change this to your target IP**

**ports\_to\_scan = [22, 80, 443, 8080]**

**# Run socket-based port scan**

**scan\_ports(target\_ip, ports\_to\_scan)**

**# Run nmap-based port scan**

**nmap\_scan(target\_ip)**

1. **Write a Java program to extract and display IP addresses, subnet masks, and default gateways from system network settings.**

**Intsatll -=** java -version

**import socket**

**import os**

**import platform**

**import subprocess**

**import re**

**def get\_ip\_addresses():**

**"""Retrieves and displays network interfaces and IP addresses."""**

**print("\n=== Network Interfaces and IP Addresses ===\n")**

**host\_name = socket.gethostname()**

**ip\_address = socket.gethostbyname(host\_name)**

**print(f"Hostname: {host\_name}")**

**print(f"IP Address: {ip\_address}\n")**

**def get\_network\_config():**

**"""Extracts subnet masks and default gateways from system settings."""**

**os\_name = platform.system().lower()**

**if "windows" in os\_name:**

**command = "ipconfig"**

**else:**

**command = "ifconfig" if os\_name == "linux" else "ip addr show"**

**try:**

**output = subprocess.check\_output(command, shell=True, text=True)**

**print("\n=== Network Configuration ===\n")**

**if "windows" in os\_name:**

**subnet\_pattern = r"Subnet Mask\s\*:\s\*([\d\.]+)"**

**gateway\_pattern = r"Default Gateway\s\*:\s\*([\d\.]+)"**

**else:**

**subnet\_pattern = r"Mask:\s\*([\d\.]+)|netmask\s\*([\d\.]+)"**

**gateway\_pattern = r"gateway\s\*([\d\.]+)"**

**subnet\_masks = re.findall(subnet\_pattern, output)**

**gateways = re.findall(gateway\_pattern, output)**

**if subnet\_masks:**

**print("Subnet Masks:")**

**for mask in subnet\_masks:**

**print(f" {mask[0] if mask[0] else mask[1]}")**

**if gateways:**

**print("\nDefault Gateways:")**

**for gateway in gateways:**

**print(f" {gateway[0]}")**

**except subprocess.CalledProcessError as e:**

**print(f"Error fetching network details: {e}")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**get\_ip\_addresses()**

**get\_network\_config()**

1. **Implement a Python script to log all incoming and outgoing network connections using**

**psutil and save the data for analysis.**

**Install -=** pip install psutil

**import psutil**

**import time**

**import csv**

**import os**

**def log\_connections(log\_file="network\_connections.csv"):**

**"""Logs all active network connections to a CSV file."""**

**# Create or open the log file**

**file\_exists = os.path.exists(log\_file)**

**with open(log\_file, mode='a', newline='') as file:**

**writer = csv.writer(file)**

**# Write the header only if the file is newly created**

**if not file\_exists:**

**writer.writerow(["Timestamp", "PID", "Process Name", "Local Address", "Remote Address", "Status"])**

**while True:**

**for conn in psutil.net\_connections(kind='inet'):**

**try:**

**# Format local and remote addresses**

**laddr = f"{conn.laddr.ip}:{conn.laddr.port}" if conn.laddr else "N/A"**

**raddr = f"{conn.raddr.ip}:{conn.raddr.port}" if conn.raddr else "N/A"**

**# Get process name**

**process\_name = psutil.Process(conn.pid).name() if conn.pid else "N/A"**

**# Write to CSV**

**writer.writerow([**

**time.strftime("%Y-%m-%d %H:%M:%S"),**

**conn.pid,**

**process\_name,**

**laddr,**

**raddr,**

**conn.status**

**])**

**except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):**

**continue**

**print("Logged network connections... (Press Ctrl+C to stop)")**

**time.sleep(5) # Log every 5 seconds**

**if \_\_name\_\_ == "\_\_main\_\_":**

**log\_connections()**

1. **Develop a Python-based monitoring tool to detect unauthorized changes to system network configuration files (/etc/network/interfaces, /etc/resolv.conf).**

**import hashlib**

**import time**

**import os**

**# List of files to monitor**

**FILES\_TO\_MONITOR = ["/etc/network/interfaces", "/etc/resolv.conf"]**

**def get\_file\_hash(filepath):**

**"""Returns the SHA256 hash of the file contents."""**

**try:**

**with open(filepath, 'rb') as file:**

**file\_contents = file.read()**

**return hashlib.sha256(file\_contents).hexdigest()**

**except FileNotFoundError:**

**return None**

**except PermissionError:**

**print(f"[ERROR] Permission denied: Cannot access {filepath}")**

**return None**

**def monitor\_files(interval=10):**

**"""Continuously monitors the files for unauthorized changes."""**

**file\_hashes = {file: get\_file\_hash(file) for file in FILES\_TO\_MONITOR}**

**print("Monitoring started. Press Ctrl+C to stop.")**

**try:**

**while True:**

**time.sleep(interval)**

**for file in FILES\_TO\_MONITOR:**

**new\_hash = get\_file\_hash(file)**

**if new\_hash is None:**

**print(f"[ALERT] {file} has been deleted or is inaccessible!")**

**elif new\_hash != file\_hashes[file]:**

**print(f"[WARNING] Unauthorized change detected in {file}!")**

**file\_hashes[file] = new\_hash # Update stored hash**

**except KeyboardInterrupt:**

**print("Monitoring stopped.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**monitor\_files()**

1. **Develop a Python script to encrypt and decrypt files before transferring them over FTP using PyCryptodome.**

**File = pip install pycryptodome  
  
from Crypto.Cipher import AES**

**from ftplib import FTP**

**import os**

**# AES Padding (PKCS7)**

**def pad(data):**

**padding\_length = 16 - (len(data) % 16)**

**return data + bytes([padding\_length] \* padding\_length)**

**def unpad(data):**

**return data[:-data[-1]]**

**# Ensure file exists**

**def ensure\_file\_exists(filename, default\_content="This is a test file for encryption."):**

**if not os.path.exists(filename):**

**with open(filename, 'w') as f:**

**f.write(default\_content)**

**print(f"[INFO] Created {filename} with default content.")**

**# AES Encryption**

**def encrypt\_file(input\_file, output\_file, key):**

**ensure\_file\_exists(input\_file) # Ensure input file exists**

**cipher = AES.new(key, AES.MODE\_CBC, iv=b"1234567890123456") # 16-byte IV**

**with open(input\_file, 'rb') as f:**

**plaintext = f.read()**

**ciphertext = cipher.encrypt(pad(plaintext))**

**with open(output\_file, 'wb') as f:**

**f.write(cipher.iv + ciphertext) # Store IV in file**

**print(f"[SUCCESS] Encrypted {input\_file} → {output\_file}")**

**# AES Decryption**

**def decrypt\_file(input\_file, output\_file, key):**

**if not os.path.exists(input\_file):**

**print(f"[ERROR] File {input\_file} not found. Skipping decryption.")**

**return**

**with open(input\_file, 'rb') as f:**

**iv = f.read(16) # Read IV from file**

**ciphertext = f.read()**

**cipher = AES.new(key, AES.MODE\_CBC, iv=iv)**

**plaintext = unpad(cipher.decrypt(ciphertext))**

**with open(output\_file, 'wb') as f:**

**f.write(plaintext)**

**print(f"[SUCCESS] Decrypted {input\_file} → {output\_file}")**

**# FTP Upload**

**def upload\_file(ftp\_server, username, password, file\_path):**

**try:**

**ftp = FTP(ftp\_server)**

**ftp.login(user=username, passwd=password)**

**with open(file\_path, 'rb') as f:**

**ftp.storbinary(f'STOR {os.path.basename(file\_path)}', f)**

**ftp.quit()**

**print(f"[SUCCESS] Uploaded {file\_path} to FTP server.")**

**except Exception as e:**

**print(f"[ERROR] Upload failed: {e}")**

**# FTP Download**

**def download\_file(ftp\_server, username, password, remote\_filename, local\_path):**

**try:**

**ftp = FTP(ftp\_server)**

**ftp.login(user=username, passwd=password)**

**with open(local\_path, 'wb') as f:**

**ftp.retrbinary(f'RETR {remote\_filename}', f.write)**

**ftp.quit()**

**print(f"[SUCCESS] Downloaded {remote\_filename} from FTP server.")**

**except Exception as e:**

**print(f"[ERROR] Download failed: {e}")**

**# Main Execution**

**if \_\_name\_\_ == "\_\_main\_\_":**

**key = b"thisisaverysecurekey!"[:16] # Ensure 16-byte key**

**ftp\_server = "ftp.example.com"**

**username = "user"**

**password = "password"**

**# Encrypt and Upload**

**encrypt\_file("sample.txt", "sample\_encrypted.bin", key)**

**upload\_file(ftp\_server, username, password, "sample\_encrypted.bin")**

**# Download and Decrypt**

**download\_file(ftp\_server, username, password, "sample\_encrypted.bin", "downloaded\_encrypted.bin")**

**decrypt\_file("downloaded\_encrypted.bin", "sample\_decrypted.txt", key)**

1. **Implement a Python-based SSH brute-force attack detection system using paramiko to log failed login attempts.**

**pip3 install paramiko**

**import paramiko**

**import socket**

**import logging**

**import time**

**# Configure logging**

**logging.basicConfig(**

**filename="ssh\_brute\_force.log",**

**level=logging.INFO,**

**format="%(asctime)s - %(levelname)s - %(message)s",**

**)**

**def detect\_brute\_force\_attempts(server\_ip, port, username, password\_list, delay=2):**

**"""**

**Detects failed SSH login attempts and logs them.**

**:param server\_ip: SSH server IP**

**:param port: SSH port (default: 22)**

**:param username: SSH username**

**:param password\_list: List of passwords to test**

**:param delay: Delay between attempts to avoid triggering IDS/IPS**

**"""**

**for password in password\_list:**

**client = paramiko.SSHClient()**

**client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())**

**try:**

**# Attempt SSH connection**

**client.connect(server\_ip, port=port, username=username, password=password, timeout=5)**

**print(f"[SUCCESS] Login successful with password: {password}")**

**logging.info(f"Successful login for {username} with password: {password}")**

**client.close()**

**break # Stop after successful login**

**except paramiko.AuthenticationException:**

**print(f"[FAILED] Login failed for {username} with password: {password}")**

**logging.warning(f"Failed login attempt for {username} with password: {password}")**

**except paramiko.SSHException as e:**

**print(f"[ERROR] SSH error: {e}")**

**logging.error(f"SSH error encountered: {e}")**

**except socket.timeout:**

**print(f"[ERROR] Connection to {server\_ip} timed out.")**

**logging.error(f"Connection to {server\_ip} timed out.")**

**except socket.error as e:**

**print(f"[ERROR] Connection error: {e}")**

**logging.error(f"Connection error: {e}")**

**break # Stop if server is unreachable**

**finally:**

**client.close() # Ensure client is closed**

**time.sleep(delay) # Delay to avoid triggering security measures**

**# Example usage**

**if \_\_name\_\_ == "\_\_main\_\_":**

**server\_ip = "192.168.1.1" # Change to actual SSH server IP**

**port = 22**

**username = "admin"**

**password\_list = ["password123", "admin123", "letmein", "123456"]**

**print(f"[\*] Starting SSH brute-force detection on {server\_ip}:{port}")**

**detect\_brute\_force\_attempts(server\_ip, port, username, password\_list)**

**print("[\*] Scan completed. Check ssh\_brute\_force.log for details.")**

1. **Write a Java program to implement a secure login system using Java Security Manager policies.**

**import java.io.FileWriter;**

**import java.io.IOException;**

**import java.util.Scanner;**

**public class SecureLogin {**

**public static void main(String[] args) {**

**String correctUsername = "admin";**

**String correctPassword = "password123";**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter username: ");**

**String username = scanner.nextLine();**

**System.out.print("Enter password: ");**

**String password = scanner.nextLine();**

**if (username.equals(correctUsername) && password.equals(correctPassword)) {**

**System.out.println("Login Successful!");**

**// Try to write login activity to file**

**try (FileWriter writer = new FileWriter("login.txt", true)) {**

**writer.write("User " + username + " logged in successfully.\n");**

**System.out.println("Login recorded in file.");**

**} catch (IOException e) {**

**System.out.println("Error writing to log file: " + e.getMessage());**

**}**

**} else {**

**System.out.println("Invalid Credentials!");**

**}**

**scanner.close();**

**}**

**}**

1. **Write a Java-based HTTPS server using Java's SSL libraries to encrypt client-server communication.**
2. **Implement AES encryption in a Java application to securely store and retrieve sensitive data.**

**import javax.crypto.Cipher;**

**import javax.crypto.KeyGenerator;**

**import javax.crypto.SecretKey;**

**import javax.crypto.spec.SecretKeySpec;**

**import java.util.Base64;**

**public class AESExample {**

**// Generate AES Key (128-bit)**

**public static SecretKey generateKey() throws Exception {**

**KeyGenerator keyGen = KeyGenerator.getInstance("AES");**

**keyGen.init(128); // 128-bit AES**

**return keyGen.generateKey();**

**}**

**// Encrypt data**

**public static String encrypt(String plainText, SecretKey secretKey) throws Exception {**

**Cipher cipher = Cipher.getInstance("AES"); // Default: ECB mode, PKCS5Padding**

**cipher.init(Cipher.ENCRYPT\_MODE, secretKey);**

**byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());**

**return Base64.getEncoder().encodeToString(encryptedBytes); // Encode to Base64**

**}**

**// Decrypt data**

**public static String decrypt(String encryptedText, SecretKey secretKey) throws Exception {**

**Cipher cipher = Cipher.getInstance("AES");**

**cipher.init(Cipher.DECRYPT\_MODE, secretKey);**

**byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));**

**return new String(decryptedBytes);**

**}**

**public static void main(String[] args) {**

**try {**

**System.out.println("Secure AES Encryption Example\n------------------------------------------");**

**// 1. Generate AES key**

**SecretKey secretKey = generateKey();**

**// 2. Data to encrypt**

**String originalData = "Sensitive User Password 123!";**

**// 3. Encrypt**

**String encrypted = encrypt(originalData, secretKey);**

**System.out.println("Original: " + originalData);**

**System.out.println("Encrypted: " + encrypted);**

**// 4. Decrypt**

**String decrypted = decrypt(encrypted, secretKey);**

**System.out.println("Decrypted: " + decrypted);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

1. **Create a Java application that verifies the integrity of a downloaded file using a SHA- 256 checksum.  
     
   NOT WORKING**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.security.MessageDigest;**

**import java.util.Formatter;**

**import java.util.Scanner;**

**public class FileChecksum {**

**public static String calculateSHA256(File file) throws Exception {**

**MessageDigest digest = MessageDigest.getInstance("SHA-256");**

**FileInputStream fis = new FileInputStream(file);**

**byte[] byteArray = new byte[1024];**

**int bytesRead;**

**while ((bytesRead = fis.read(byteArray)) != -1) {**

**digest.update(byteArray, 0, bytesRead);**

**}**

**fis.close();**

**byte[] hashBytes = digest.digest();**

**return bytesToHex(hashBytes);**

**}**

**private static String bytesToHex(byte[] bytes) {**

**Formatter formatter = new Formatter();**

**for (byte b : bytes) {**

**formatter.format("%02x", b);**

**}**

**String result = formatter.toString();**

**formatter.close();**

**return result;**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the full path to the file: ");**

**String filePath = scanner.nextLine();**

**scanner.close();**

**try {**

**File file = new File(filePath);**

**if (!file.exists()) {**

**System.out.println("File not found: " + file.getAbsolutePath());**

**return;**

**}**

**String checksum = calculateSHA256(file);**

**System.out.println("SHA-256 Checksum: " + checksum);**

**} catch (Exception e) {**

**System.err.println("Error computing checksum:");**

**e.printStackTrace();**

**}**

**}**

**}**

1. **Develop a Java program that implements RSA encryption and decryption for secure data transfer.**

**import java.security.KeyPair;**

**import java.security.KeyPairGenerator;**

**import java.security.PrivateKey;**

**import java.security.PublicKey;**

**import javax.crypto.Cipher;**

**import java.util.Base64;**

**public class RSAEncryption {**

**// Generate RSA key pair**

**private static KeyPair generateKeyPair() throws Exception {**

**KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");**

**keyPairGenerator.initialize(2048); // 2048-bit key size**

**return keyPairGenerator.generateKeyPair();**

**}**

**// Encrypt data with public key**

**public static String encrypt(String data, PublicKey publicKey) throws Exception {**

**Cipher cipher = Cipher.getInstance("RSA");**

**cipher.init(Cipher.ENCRYPT\_MODE, publicKey);**

**byte[] encryptedBytes = cipher.doFinal(data.getBytes());**

**return Base64.getEncoder().encodeToString(encryptedBytes);**

**}**

**// Decrypt data with private key**

**public static String decrypt(String encryptedData, PrivateKey privateKey) throws Exception {**

**Cipher cipher = Cipher.getInstance("RSA");**

**cipher.init(Cipher.DECRYPT\_MODE, privateKey);**

**byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedData));**

**return new String(decryptedBytes);**

**}**

**// Main method to demonstrate encryption and decryption**

**public static void main(String[] args) {**

**try {**

**KeyPair keyPair = generateKeyPair();**

**String originalData = "SecureMessage123";**

**// Encrypt the original message**

**String encryptedData = encrypt(originalData, keyPair.getPublic());**

**System.out.println("Encrypted Data: " + encryptedData);**

**// Decrypt the message**

**String decryptedData = decrypt(encryptedData, keyPair.getPrivate());**

**System.out.println("Decrypted Data: " + decryptedData);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

1. **Write a Java-based secure authentication system that uses hashed passwords with salting for user login.**

**import java.security.SecureRandom;**

**import java.security.MessageDigest;**

**import java.security.NoSuchAlgorithmException;**

**import java.util.Base64;**

**import java.util.HashMap;**

**public class SecureAuthentication {**

**private static final SecureRandom RANDOM = new SecureRandom();**

**private static final HashMap<String, String[]> userDatabase = new HashMap<>();**

**// Generate a random salt**

**public static String generateSalt() {**

**byte[] salt = new byte[16];**

**RANDOM.nextBytes(salt);**

**return Base64.getEncoder().encodeToString(salt);**

**}**

**// Hash the password with the salt**

**public static String hashPassword(String password, String salt) throws NoSuchAlgorithmException {**

**MessageDigest md = MessageDigest.getInstance("SHA-256");**

**md.update(salt.getBytes());**

**byte[] hashedPassword = md.digest(password.getBytes());**

**return Base64.getEncoder().encodeToString(hashedPassword);**

**}**

**// Register a new user**

**public static void registerUser(String username, String password) throws Exception {**

**String salt = generateSalt();**

**String hashedPassword = hashPassword(password, salt);**

**userDatabase.put(username, new String[]{hashedPassword, salt});**

**System.out.println("User registered: " + username);**

**}**

**// Authenticate user credentials**

**public static boolean authenticateUser(String username, String password) throws Exception {**

**if (userDatabase.containsKey(username)) {**

**String[] storedData = userDatabase.get(username);**

**String storedHash = storedData[0];**

**String salt = storedData[1];**

**String attemptedHash = hashPassword(password, salt);**

**return storedHash.equals(attemptedHash);**

**}**

**return false;**

**}**

**// Main method to test registration and authentication**

**public static void main(String[] args) throws Exception {**

**registerUser("user1", "securePassword");**

**boolean success = authenticateUser("user1", "securePassword");**

**System.out.println("Login success: " + success);**

**boolean fail = authenticateUser("user1", "wrongPassword");**

**System.out.println("Login with wrong password: " + fail);**

**}**

**}**

1. **Implement and compare hash functions (MD5, SHA-256, SHA-512) using a Python program to verify file integrity.**

**import hashlib**

**import os**

**# Function to compute hash values for a given file using MD5, SHA-256, and SHA-512**

**def compute\_hash(file\_path):**

**hashes = {**

**"MD5": hashlib.md5(),**

**"SHA-256": hashlib.sha256(),**

**"SHA-512": hashlib.sha512(),**

**}**

**try:**

**with open(file\_path, "rb") as file:**

**# Read in chunks (1MB) to handle large files efficiently**

**while chunk := file.read(1024 \* 1024):**

**for algo in hashes.values():**

**algo.update(chunk)**

**# Convert hash objects to hexadecimal strings**

**return {name: algo.hexdigest() for name, algo in hashes.items()}**

**except FileNotFoundError:**

**print("[ERROR] File not found. Please enter a valid file path.")**

**return None**

**except Exception as e:**

**print(f"[ERROR] An error occurred: {e}")**

**return None**

**# Function to compare and display hash values**

**def compare\_hashes(hashes):**

**if hashes:**

**print("\nFile Hash Values:")**

**print("=" \* 50)**

**for algo, hash\_value in hashes.items():**

**print(f"{algo}: {hash\_value}")**

**print("=" \* 50)**

**# Main function**

**if \_\_name\_\_ == "\_\_main\_\_":**

**file\_path = input("Enter the path of the file to hash: ").strip()**

**if os.path.isfile(file\_path): # Check if file exists**

**hashes = compute\_hash(file\_path)**

**compare\_hashes(hashes)**

**else:**

**print("[ERROR] Invalid file path. Please check and try again.")**

1. **Write a Python program to generate an HMAC-based authentication system using the HMAC module.**

**import hmac**

**import hashlib**

**import os**

**# Securely generate a random key (For real-world use, store securely)**

**SECRET\_KEY = b'supersecretkey'**

**# Function to generate HMAC for a given message**

**def generate\_hmac(message: str, key: bytes) -> str:**

**return hmac.new(key, message.encode(), hashlib.sha256).hexdigest()**

**# Function to verify the HMAC for a given message**

**def verify\_hmac(message: str, key: bytes, expected\_hmac: str) -> bool:**

**generated\_hmac = generate\_hmac(message, key)**

**return hmac.compare\_digest(generated\_hmac, expected\_hmac)**

**# Simulating user authentication process**

**def authenticate\_user(username: str, password: str) -> bool:**

**# Simulated stored password hash (In real-world, store in a database)**

**stored\_password\_hash = generate\_hmac("userpassword", SECRET\_KEY)**

**# Verify the provided password**

**provided\_password\_hash = generate\_hmac(password, SECRET\_KEY)**

**if verify\_hmac(provided\_password\_hash, SECRET\_KEY, stored\_password\_hash):**

**print("\n✅ Authentication Successful! Welcome,", username)**

**return True**

**else:**

**print("\n❌ Authentication Failed! Invalid credentials.")**

**return False**

**# Main function: Simulating login attempt**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("\n🔒 Welcome to the Secure Authentication System!\n")**

**username = input("Enter username: ").strip()**

**password = input("Enter password: ").strip()**

**# Authenticate the user**

**authenticate\_user(username, password)**

1. **Develop a Python script to create and verify digital signatures using the PyCryptodome library**

**from Crypto.PublicKey import RSA**

**from Crypto.Signature import pkcs1\_15**

**from Crypto.Hash import SHA256**

**import binascii**

**# Function to generate RSA key pair**

**def generate\_keys():**

**key = RSA.generate(2048) # 2048-bit RSA key for security**

**private\_key = key.export\_key()**

**public\_key = key.publickey().export\_key()**

**return private\_key, public\_key**

**# Function to create a digital signature**

**def create\_signature(message: str, private\_key: bytes) -> bytes:**

**h = SHA256.new(message.encode()) # Hash the message using SHA-256**

**private\_key\_obj = RSA.import\_key(private\_key) # Load the private key**

**signature = pkcs1\_15.new(private\_key\_obj).sign(h) # Sign the hash**

**return signature**

**# Function to verify the digital signature**

**def verify\_signature(message: str, signature: bytes, public\_key: bytes):**

**h = SHA256.new(message.encode()) # Hash the message**

**public\_key\_obj = RSA.import\_key(public\_key) # Load the public key**

**try:**

**pkcs1\_15.new(public\_key\_obj).verify(h, signature)**

**print("\n✅ Signature is VALID! Message is authentic.")**

**except (ValueError, TypeError):**

**print("\n❌ Signature verification FAILED! Message integrity compromised.")**

**# Main function**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("\n🔐 Digital Signature System using PyCryptodome\n")**

**# Step 1: Generate RSA keys**

**private\_key, public\_key = generate\_keys()**

**# Step 2: Define a message**

**message = "This is a secret message!"**

**# Step 3: Create a digital signature**

**print(f"Original Message: {message}")**

**signature = create\_signature(message, private\_key)**

**# Print the signature in hexadecimal format**

**print(f"🔑 Digital Signature: {binascii.hexlify(signature).decode()}")**

**# Step 4: Verify the digital signature**

**print("\n🔍 Verifying the signature with the public key...")**

**verify\_signature(message, signature, public\_key)**

**# Step 5: Modify the message and test again**

**tampered\_message = "This is a tampered message!"**

**print("\n⚠️ Testing with a modified message...")**

**verify\_signature(tampered\_message, signature, public\_key)**

1. **Implement a challenge-response authentication system in Python using one-time passwords (OTP) and secrets module.**

**import secrets**

**import hmac**

**import hashlib**

**import time**

**# Secret key for OTP generation (should be stored securely in real-world applications)**

**SECRET\_KEY = b"SuperSecretKey"**

**# OTP Expiry time (in seconds)**

**OTP\_EXPIRY = 30**

**# Function to generate a one-time password (OTP) using HMAC-SHA256**

**def generate\_otp(challenge: str, secret\_key: bytes) -> str:**

**"""**

**Generates a deterministic OTP using HMAC-SHA256 based on a challenge and secret key.**

**"""**

**hmac\_obj = hmac.new(secret\_key, challenge.encode(), hashlib.sha256)**

**otp\_hex = hmac\_obj.hexdigest()[:8] # Take the first 8 hex characters**

**return otp\_hex.upper()**

**# Server-side function to generate a challenge and corresponding OTP**

**def server\_generate\_challenge():**

**challenge = secrets.token\_hex(4) # Generate a random 4-byte challenge (8 hex characters)**

**otp = generate\_otp(challenge, SECRET\_KEY) # Generate the OTP**

**timestamp = int(time.time()) # Capture the timestamp**

**return challenge, otp, timestamp**

**# Client-side function to generate OTP response based on the received challenge**

**def client\_generate\_otp(challenge: str):**

**"""**

**The client generates an OTP using the challenge and the same secret key.**

**"""**

**return generate\_otp(challenge, SECRET\_KEY)**

**# Function to verify OTP with a time-based expiry**

**def verify\_otp(server\_otp: str, client\_otp: str, timestamp: int) -> bool:**

**"""**

**Validates the OTP and ensures it has not expired.**

**"""**

**current\_time = int(time.time())**

**if current\_time - timestamp > OTP\_EXPIRY:**

**print("\n⏳ OTP has expired! Please request a new one.")**

**return False**

**if hmac.compare\_digest(server\_otp, client\_otp):**

**print("\n✅ Authentication Successful!")**

**return True**

**else:**

**print("\n❌ Authentication Failed! Incorrect OTP.")**

**return False**

**# Main function to simulate challenge-response authentication**

**def authenticate\_user():**

**print("\n🔐 Welcome to the Secure OTP Authentication System!\n")**

**# Step 1: Server generates a challenge and OTP**

**challenge, correct\_otp, timestamp = server\_generate\_challenge()**

**# Step 2: Server sends challenge to the client**

**print(f"🛡️ Challenge (sent to client): {challenge}")**

**# Step 3: Client generates OTP response**

**client\_otp = client\_generate\_otp(challenge)**

**# Step 4: Server verifies OTP**

**print("\n🔍 Verifying OTP...")**

**verify\_otp(correct\_otp, client\_otp, timestamp)**

**# Run authentication process**

**if \_\_name\_\_ == "\_\_main\_\_":**

**authenticate\_user()**

1. **Develop a Python script to implement two-factor authentication (2FA) using the pyotp**

**library.  
file = pip install pyotp  
  
  
import pyotp**

**import time**

**# Function to generate a secret key for 2FA**

**def generate\_secret():**

**"""**

**Generates a random secret key for the user.**

**This key is shared between the client and server.**

**"""**

**secret = pyotp.random\_base32() # Generate a base32-encoded secret key**

**print(f"🔑 Your secret key (store this securely!): {secret}")**

**return secret**

**# Function to generate an OTP (this is typically done by the client)**

**def generate\_otp(secret):**

**"""**

**Generates a one-time password (OTP) using the shared secret key.**

**"""**

**totp = pyotp.TOTP(secret)**

**otp = totp.now() # Generate current OTP**

**print(f"📲 Generated OTP: {otp}")**

**return otp**

**# Function to verify OTP (this is typically done by the server)**

**def verify\_otp(secret, otp):**

**"""**

**Verifies the OTP entered by the user.**

**"""**

**totp = pyotp.TOTP(secret)**

**if totp.verify(otp):**

**print("✅ OTP Verified Successfully! Authentication Successful.")**

**else:**

**print("❌ Invalid OTP! Verification Failed.")**

**# Main function to run the 2FA authentication process**

**def main():**

**print("\n🔐 Two-Factor Authentication (2FA) Demo\n")**

**# Step 1: Generate a secret key (this would be stored securely in a real system)**

**secret = generate\_secret()**

**# Step 2: Generate an OTP (client generates OTP based on the shared secret)**

**otp = generate\_otp(secret)**

**# Step 3: User enters the OTP (simulating client input)**

**entered\_otp = input("\n🔢 Enter the OTP: ").strip()**

**# Step 4: Verify the OTP (server-side verification)**

**verify\_otp(secret, entered\_otp)**

**# Run the script**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

1. **Write a Python program to compare the performance and security of SHA-256 and bcrypt hashing algorithms.**

**import hashlib**

**import bcrypt**

**import time**

**# Function to hash using SHA-256**

**def hash\_sha256(data):**

**"""Hashes the given data using SHA-256."""**

**sha256 = hashlib.sha256()**

**sha256.update(data.encode('utf-8'))**

**return sha256.hexdigest()**

**# Function to hash using bcrypt**

**def hash\_bcrypt(data):**

**"""Hashes the given data using bcrypt with a randomly generated salt."""**

**salt = bcrypt.gensalt(rounds=12) # Adjust cost factor (higher = more secure, but slower)**

**return bcrypt.hashpw(data.encode('utf-8'), salt)**

**# Function to compare performance and security of SHA-256 and bcrypt**

**def compare\_hashing\_algorithms():**

**data = "supersecretpassword"**

**print("\n🔐 Hashing Performance & Security Comparison\n")**

**# SHA-256 hashing**

**start\_time = time.time()**

**sha256\_hash = hash\_sha256(data)**

**sha256\_time = time.time() - start\_time**

**sha256\_time = max(sha256\_time, 1e-6) # Prevent division by zero**

**print(f"SHA-256 Hash: {sha256\_hash}")**

**print(f"SHA-256 Time: {sha256\_time:.6f} seconds")**

**# bcrypt hashing**

**start\_time = time.time()**

**bcrypt\_hash = hash\_bcrypt(data)**

**bcrypt\_time = time.time() - start\_time**

**print(f"bcrypt Hash: {bcrypt\_hash.decode()}") # Convert bytes to string**

**print(f"bcrypt Time: {bcrypt\_time:.6f} seconds")**

**# Comparing performance**

**speed\_ratio = bcrypt\_time / sha256\_time**

**print("\n⏳ Performance Comparison:")**

**print(f"SHA-256 is approximately {speed\_ratio:.2f} times faster than bcrypt.")**

**# Security considerations**

**print("\n🔒 Security Considerations:")**

**print("1. SHA-256 is very fast, making it vulnerable to brute-force and rainbow table attacks.")**

**print("2. bcrypt is deliberately slow, making brute-force attacks significantly harder.")**

**print("3. bcrypt includes a unique salt for every hash, preventing rainbow table attacks.")**

**print("4. bcrypt's computational cost can be increased over time to maintain security.")**

**# Main function to run the comparison**

**if \_\_name\_\_ == "\_\_main\_\_":**

**compare\_hashing\_algorithms()**

1. **Implement a Python script to encrypt and authenticate messages using AES-GCM (Authenticated Encryption).**

**from Crypto.Cipher import AES**

**from Crypto.Random import get\_random\_bytes**

**import base64**

**# Function to generate a random AES key (AES-256 requires a 32-byte key)**

**def generate\_key():**

**return get\_random\_bytes(32) # 256-bit key for AES**

**# Function to encrypt a message using AES-GCM**

**def encrypt\_message(key, message, aad=None):**

**"""Encrypts a message using AES-GCM with an optional AAD (Additional Authenticated Data)."""**

**nonce = get\_random\_bytes(12) # Recommended size for AES-GCM nonce**

**cipher = AES.new(key, AES.MODE\_GCM, nonce=nonce)**

**# Include additional authenticated data (AAD) if provided**

**if aad:**

**cipher.update(aad)**

**ciphertext, tag = cipher.encrypt\_and\_digest(message.encode('utf-8'))**

**return ciphertext, nonce, tag**

**# Function to decrypt a message using AES-GCM and verify its authenticity**

**def decrypt\_message(key, ciphertext, nonce, tag, aad=None):**

**"""Decrypts a message and verifies its authenticity using AES-GCM."""**

**cipher = AES.new(key, AES.MODE\_GCM, nonce=nonce)**

**# Include AAD if provided**

**if aad:**

**cipher.update(aad)**

**try:**

**decrypted\_message = cipher.decrypt\_and\_verify(ciphertext, tag)**

**return decrypted\_message.decode('utf-8')**

**except (ValueError, TypeError):**

**print("❌ Decryption failed! The message may have been tampered with.")**

**return None**

**# Main function to demonstrate AES-GCM encryption and decryption**

**def main():**

**message = "This is a confidential and authenticated message."**

**# Generate AES key**

**key = generate\_key()**

**# Additional Authenticated Data (AAD) - Optional**

**aad = b"Authenticated Data"**

**# Encrypt the message**

**print("\n🔒 Encrypting message...")**

**ciphertext, nonce, tag = encrypt\_message(key, message, aad)**

**# Print encryption details in Base64 format for readability**

**print(f"Ciphertext (Base64): {base64.b64encode(ciphertext).decode()}")**

**print(f"Nonce (Base64): {base64.b64encode(nonce).decode()}")**

**print(f"Authentication Tag (Base64): {base64.b64encode(tag).decode()}")**

**# Decrypt the message**

**print("\n🔑 Decrypting message...")**

**decrypted\_message = decrypt\_message(key, ciphertext, nonce, tag, aad)**

**if decrypted\_message:**

**print(f"✅ Decrypted Message: {decrypted\_message}")**

**else:**

**print("❌ Failed to decrypt the message.")**

**# Run the main function**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

1. **Develop a Python program to simulate a ransomware attack by encrypting files in a folder and then decrypting them with a key.**

**import os**

**from cryptography.fernet import Fernet**

**# Function to generate and save an encryption key**

**def generate\_key():**

**key = Fernet.generate\_key()**

**with open("encryption\_key.key", "wb") as key\_file:**

**key\_file.write(key)**

**return key**

**# Function to load the encryption key**

**def load\_key():**

**if not os.path.exists("encryption\_key.key"):**

**print("❌ Error: Encryption key not found. Please encrypt files first!")**

**return None**

**return open("encryption\_key.key", "rb").read()**

**# Function to encrypt all files in a specified folder**

**def encrypt\_files(folder\_path, key):**

**cipher = Fernet(key)**

**for file\_name in os.listdir(folder\_path):**

**file\_path = os.path.join(folder\_path, file\_name)**

**if os.path.isfile(file\_path):**

**with open(file\_path, "rb") as file:**

**file\_data = file.read()**

**encrypted\_data = cipher.encrypt(file\_data)**

**with open(file\_path, "wb") as file:**

**file.write(encrypted\_data)**

**print("✅ All files have been encrypted.")**

**# Function to decrypt all files in a specified folder**

**def decrypt\_files(folder\_path, key):**

**cipher = Fernet(key)**

**for file\_name in os.listdir(folder\_path):**

**file\_path = os.path.join(folder\_path, file\_name)**

**if os.path.isfile(file\_path):**

**with open(file\_path, "rb") as file:**

**encrypted\_data = file.read()**

**decrypted\_data = cipher.decrypt(encrypted\_data)**

**with open(file\_path, "wb") as file:**

**file.write(decrypted\_data)**

**print("✅ All files have been decrypted.")**

**# Main function to handle user input**

**def main():**

**folder = "test\_folder" # Change this to your target folder**

**os.makedirs(folder, exist\_ok=True) # Ensure folder exists**

**option = input("Enter 'E' to encrypt or 'D' to decrypt: ").strip().upper()**

**if option == 'E':**

**key = generate\_key()**

**encrypt\_files(folder, key)**

**elif option == 'D':**

**key = load\_key()**

**if key:**

**decrypt\_files(folder, key)**

**else:**

**print("❌ Invalid option. Use 'E' for encrypt or 'D' for decrypt.")**

**# Run the script**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

1. **Write a Python-based firewall rule tester that sends test packets and analyzes blocked/allowed responses.**
2. **Perform encryption, decryption using the following substitution techniques**
   1. **Ceaser cipher**
   2. **Playfair cipher**
   3. **Hill Cipher**
   4. **Vigenere cipher**

**a=**

**def caesar\_cipher\_encrypt(text, shift):**

**"""Encrypts text using Caesar Cipher with given shift."""**

**result = ""**

**for char in text:**

**if char.isalpha(): # Only shift letters**

**shift\_base = ord('A') if char.isupper() else ord('a')**

**result += chr((ord(char) - shift\_base + shift) % 26 + shift\_base)**

**else:**

**result += char # Keep spaces, punctuation unchanged**

**return result**

**def caesar\_cipher\_decrypt(text, shift):**

**"""Decrypts text using Caesar Cipher with given shift."""**

**return caesar\_cipher\_encrypt(text, -shift) # Reverse the shift**

**# Main function to demonstrate the cipher**

**if \_\_name\_\_ == "\_\_main\_\_":**

**message = "Anna University"**

**shift = 3**

**print("Simulating Caesar Cipher")**

**print("-" \* 50)**

**print(f"Input Message : {message}")**

**encrypted\_message = caesar\_cipher\_encrypt(message, shift)**

**print(f"Encrypted Message: {encrypted\_message}")**

**decrypted\_message = caesar\_cipher\_decrypt(encrypted\_message, shift)**

**print(f"Decrypted Message: {decrypted\_message}")  
  
  
b=  
  
import numpy as np**

**def prepare\_text(text, replace\_j=True):**

**"""Prepares text by converting to uppercase, removing non-letters, and replacing 'J' with 'I' if needed."""**

**text = text.upper().replace("J", "I") if replace\_j else text.upper()**

**text = ''.join(filter(str.isalpha, text))**

**prepared\_text = ""**

**i = 0**

**while i < len(text):**

**prepared\_text += text[i]**

**if i + 1 < len(text) and text[i] == text[i + 1]: # Duplicate letters in digram**

**prepared\_text += 'X'**

**elif i + 1 >= len(text): # Odd-length text, append 'X'**

**prepared\_text += 'X'**

**else:**

**prepared\_text += text[i + 1]**

**i += 2**

**return prepared\_text**

**def create\_playfair\_matrix(key):**

**"""Creates a 5x5 Playfair cipher key table."""**

**key = prepare\_text(key, replace\_j=True) + "ABCDEFGHIKLMNOPQRSTUVWXYZ"**

**matrix = []**

**seen = set()**

**for char in key:**

**if char not in seen:**

**seen.add(char)**

**matrix.append(char)**

**return np.array(matrix).reshape(5, 5)**

**def find\_position(matrix, char):**

**"""Finds row and column of a character in the Playfair matrix."""**

**for row in range(5):**

**for col in range(5):**

**if matrix[row][col] == char:**

**return row, col**

**return None**

**def playfair\_encrypt(text, matrix):**

**"""Encrypts text using the Playfair cipher."""**

**encrypted\_text = ""**

**text = prepare\_text(text)**

**for i in range(0, len(text), 2):**

**a, b = text[i], text[i + 1]**

**row\_a, col\_a = find\_position(matrix, a)**

**row\_b, col\_b = find\_position(matrix, b)**

**if row\_a == row\_b: # Same row: Shift right**

**encrypted\_text += matrix[row\_a][(col\_a + 1) % 5]**

**encrypted\_text += matrix[row\_b][(col\_b + 1) % 5]**

**elif col\_a == col\_b: # Same column: Shift down**

**encrypted\_text += matrix[(row\_a + 1) % 5][col\_a]**

**encrypted\_text += matrix[(row\_b + 1) % 5][col\_b]**

**else: # Rectangle rule: Swap columns**

**encrypted\_text += matrix[row\_a][col\_b]**

**encrypted\_text += matrix[row\_b][col\_a]**

**return encrypted\_text**

**def playfair\_decrypt(text, matrix):**

**"""Decrypts text using the Playfair cipher."""**

**decrypted\_text = ""**

**for i in range(0, len(text), 2):**

**a, b = text[i], text[i + 1]**

**row\_a, col\_a = find\_position(matrix, a)**

**row\_b, col\_b = find\_position(matrix, b)**

**if row\_a == row\_b: # Same row: Shift left**

**decrypted\_text += matrix[row\_a][(col\_a - 1) % 5]**

**decrypted\_text += matrix[row\_b][(col\_b - 1) % 5]**

**elif col\_a == col\_b: # Same column: Shift up**

**decrypted\_text += matrix[(row\_a - 1) % 5][col\_a]**

**decrypted\_text += matrix[(row\_b - 1) % 5][col\_b]**

**else: # Rectangle rule: Swap columns**

**decrypted\_text += matrix[row\_a][col\_b]**

**decrypted\_text += matrix[row\_b][col\_a]**

**return decrypted\_text**

**# Main function to demonstrate Playfair Cipher**

**if \_\_name\_\_ == "\_\_main\_\_":**

**key = "CSE"**

**message = "Security Lab"**

**print("Simulating Playfair Cipher")**

**print("-" \* 50)**

**playfair\_matrix = create\_playfair\_matrix(key)**

**print("Playfair Cipher Matrix:")**

**print(playfair\_matrix, "\n")**

**encrypted\_message = playfair\_encrypt(message, playfair\_matrix)**

**print(f"Input Message : {message}")**

**print(f"Encrypted Message: {encrypted\_message}")**

**decrypted\_message = playfair\_decrypt(encrypted\_message, playfair\_matrix)**

**print(f"Decrypted Message: {decrypted\_message}")  
  
  
c=**

**import numpy as np**

**def prepare\_text(text, block\_size=3):**

**"""Converts text to uppercase, removes spaces, and pads if necessary."""**

**text = text.upper().replace(" ", "")**

**while len(text) % block\_size != 0:**

**text += 'X' # Padding with 'X'**

**return text**

**def text\_to\_numbers(text):**

**"""Converts text to numerical values (A=0, B=1, ..., Z=25)."""**

**return [ord(char) - ord('A') for char in text]**

**def numbers\_to\_text(numbers):**

**"""Converts numerical values back to text."""**

**return ''.join(chr(num % 26 + ord('A')) for num in numbers)**

**def matrix\_mod\_inverse(matrix, mod=26):**

**"""Finds the modular inverse of a matrix under modulo 26."""**

**determinant = int(round(np.linalg.det(matrix))) # Determinant of the matrix**

**determinant\_inv = pow(determinant, -1, mod) # Modular inverse of determinant**

**adjugate\_matrix = np.round(determinant \* np.linalg.inv(matrix)).astype(int) % mod**

**return (determinant\_inv \* adjugate\_matrix) % mod**

**def hill\_encrypt(text, key\_matrix):**

**"""Encrypts text using Hill Cipher."""**

**text\_numbers = text\_to\_numbers(text)**

**key\_size = key\_matrix.shape[0]**

**encrypted\_numbers = []**

**for i in range(0, len(text\_numbers), key\_size):**

**block = np.array(text\_numbers[i:i+key\_size]).reshape(-1, 1)**

**encrypted\_block = np.dot(key\_matrix, block) % 26**

**encrypted\_numbers.extend(encrypted\_block.flatten())**

**return numbers\_to\_text(encrypted\_numbers)**

**def hill\_decrypt(encrypted\_text, key\_matrix):**

**"""Decrypts text using Hill Cipher."""**

**key\_matrix\_inv = matrix\_mod\_inverse(key\_matrix, 26) # Get inverse of key matrix**

**encrypted\_numbers = text\_to\_numbers(encrypted\_text)**

**key\_size = key\_matrix.shape[0]**

**decrypted\_numbers = []**

**for i in range(0, len(encrypted\_numbers), key\_size):**

**block = np.array(encrypted\_numbers[i:i+key\_size]).reshape(-1, 1)**

**decrypted\_block = np.dot(key\_matrix\_inv, block) % 26**

**decrypted\_numbers.extend(decrypted\_block.flatten())**

**return numbers\_to\_text(decrypted\_numbers)**

**# Main function to demonstrate Hill Cipher**

**if \_\_name\_\_ == "\_\_main\_\_":**

**key\_matrix = np.array([[1, 2, 1],**

**[2, 3, 2],**

**[2, 2, 1]]) # 3x3 Key matrix**

**message = "SecurityLaboratory"**

**print("Simulating Hill Cipher")**

**print("-" \* 50)**

**prepared\_message = prepare\_text(message, key\_matrix.shape[0])**

**print(f"Input Message : {message}")**

**print(f"Padded Message : {prepared\_message}")**

**encrypted\_message = hill\_encrypt(prepared\_message, key\_matrix)**

**print(f"Encrypted Message: {encrypted\_message}")**

**decrypted\_message = hill\_decrypt(encrypted\_message, key\_matrix)**

**print(f"Decrypted Message: {decrypted\_message}")  
  
d=   
  
def prepare\_text(text):**

**"""Converts text to uppercase and removes non-alphabetic characters."""**

**return ''.join(filter(str.isalpha, text.upper()))**

**def vigenere\_encrypt(plaintext, key):**

**"""Encrypts the plaintext using the Vigenère cipher."""**

**plaintext = prepare\_text(plaintext)**

**key = prepare\_text(key)**

**encrypted\_text = ""**

**key\_length = len(key)**

**for i, char in enumerate(plaintext):**

**shift = ord(key[i % key\_length]) - ord('A')**

**encrypted\_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))**

**encrypted\_text += encrypted\_char**

**return encrypted\_text**

**def vigenere\_decrypt(ciphertext, key):**

**"""Decrypts the ciphertext using the Vigenère cipher."""**

**ciphertext = prepare\_text(ciphertext)**

**key = prepare\_text(key)**

**decrypted\_text = ""**

**key\_length = len(key)**

**for i, char in enumerate(ciphertext):**

**shift = ord(key[i % key\_length]) - ord('A')**

**decrypted\_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))**

**decrypted\_text += decrypted\_char**

**return decrypted\_text**

**# Main function to demonstrate Vigenère Cipher**

**if \_\_name\_\_ == "\_\_main\_\_":**

**key = "VIGENERECIPHER"**

**message = "Security Laboratory"**

**print("Simulating Vigenère Cipher")**

**print("-" \* 50)**

**print(f"Input Message : {message}")**

**encrypted\_message = vigenere\_encrypt(message, key)**

**print(f"Encrypted Message: {encrypted\_message}")**

**decrypted\_message = vigenere\_decrypt(encrypted\_message, key)**

**print(f"Decrypted Message: {decrypted\_message}")**

1. **Perform encryption and decryption using following transposition techniques**
   1. **Rail fence**
   2. **Row & Column Transformation**

**a=  
  
  
def rail\_fence\_encrypt(text, depth):**

**"""Encrypts the text using Rail Fence Cipher."""**

**text = text.replace(" ", "") # Remove spaces for encryption**

**rail = [['\n' for \_ in range(len(text))] for \_ in range(depth)]**

**# Create the rail pattern**

**row, direction = 0, 1**

**for i, char in enumerate(text):**

**rail[row][i] = char**

**row += direction**

**if row == depth - 1 or row == 0:**

**direction \*= -1 # Change direction at top and bottom**

**# Read the rails row-wise to get ciphertext**

**encrypted\_text = ''.join(char for row in rail for char in row if char != '\n')**

**return encrypted\_text**

**def rail\_fence\_decrypt(ciphertext, depth):**

**"""Decrypts the text using Rail Fence Cipher."""**

**rail = [['\n' for \_ in range(len(ciphertext))] for \_ in range(depth)]**

**# Create the rail pattern with placeholders**

**row, direction = 0, 1**

**for i in range(len(ciphertext)):**

**rail[row][i] = '\*'**

**row += direction**

**if row == depth - 1 or row == 0:**

**direction \*= -1 # Change direction at top and bottom**

**# Replace placeholders with actual ciphertext**

**index = 0**

**for i in range(depth):**

**for j in range(len(ciphertext)):**

**if rail[i][j] == '\*' and index < len(ciphertext):**

**rail[i][j] = ciphertext[index]**

**index += 1**

**# Read message in a zigzag manner**

**row, direction = 0, 1**

**decrypted\_text = []**

**for i in range(len(ciphertext)):**

**decrypted\_text.append(rail[row][i])**

**row += direction**

**if row == depth - 1 or row == 0:**

**direction \*= -1 # Change direction at top and bottom**

**return ''.join(decrypted\_text)**

**# Main function to demonstrate Rail Fence Cipher**

**if \_\_name\_\_ == "\_\_main\_\_":**

**message = "Anna University Chennai"**

**depth = 2**

**print("Simulating Rail Fence Cipher")**

**print("-" \* 50)**

**print(f"Input Message : {message}")**

**encrypted\_message = rail\_fence\_encrypt(message, depth)**

**print(f"Encrypted Message: {encrypted\_message}")**

**decrypted\_message = rail\_fence\_decrypt(encrypted\_message, depth)**

**print(f"Decrypted Message: {decrypted\_message}")  
  
b=  
  
import math**

**def encrypt(plaintext, columns):**

**plaintext = plaintext.replace(" ", "") # Remove spaces**

**rows = math.ceil(len(plaintext) / columns) # Determine number of rows**

**grid = [['#'] \* columns for \_ in range(rows)] # Create grid with padding**

**# Fill grid row-wise**

**index = 0**

**for i in range(rows):**

**for j in range(columns):**

**if index < len(plaintext):**

**grid[i][j] = plaintext[index]**

**index += 1**

**# Read column-wise to get ciphertext**

**ciphertext = "".join(grid[i][j] for j in range(columns) for i in range(rows))**

**return ciphertext**

**def decrypt(ciphertext, columns):**

**rows = math.ceil(len(ciphertext) / columns) # Determine number of rows**

**grid = [['#'] \* columns for \_ in range(rows)] # Create grid**

**# Fill grid column-wise**

**index = 0**

**for j in range(columns):**

**for i in range(rows):**

**if index < len(ciphertext):**

**grid[i][j] = ciphertext[index]**

**index += 1**

**# Read row-wise to get the original message**

**plaintext = "".join(grid[i][j] for i in range(rows) for j in range(columns)).rstrip("#")**

**return plaintext**

**# Main function**

**if \_\_name\_\_ == "\_\_main\_\_":**

**plaintext = input("Enter the plain text: ")**

**columns = int(input("Enter the number of columns: "))**

**print("\nSimulating Row and Column Transposition Cipher")**

**print("------------------------------------------------")**

**# Encrypt**

**encrypted\_text = encrypt(plaintext, columns)**

**print("Encrypted Message:", encrypted\_text)**

**# Decrypt**

**decrypted\_text = decrypt(encrypted\_text, columns)**

**print("Decrypted Message:", decrypted\_text)**

1. **Apply DES algorithm for practical applications.**

**import javax.crypto.\*;**

**import java.security.\*;**

**import java.util.Base64;**

**public class DES {**

**public static void main(String[] args) {**

**try {**

**System.out.println("Message Encryption Using DES Algorithm");**

**System.out.println("-----------------------------------------------------------");**

**// Step 1: Generate DES key**

**KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");**

**SecretKey desKey = keyGenerator.generateKey();**

**// Step 2: Create Cipher instance**

**Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");**

**// Step 3: Convert message to byte array**

**String message = "Secret Information";**

**byte[] messageBytes = message.getBytes();**

**System.out.println("Original Message : " + message);**

**// Step 4: Encrypt**

**cipher.init(Cipher.ENCRYPT\_MODE, desKey);**

**byte[] encryptedBytes = cipher.doFinal(messageBytes);**

**String encryptedBase64 = Base64.getEncoder().encodeToString(encryptedBytes);**

**System.out.println("Encrypted Message (Base64) : " + encryptedBase64);**

**// Step 5: Decrypt**

**cipher.init(Cipher.DECRYPT\_MODE, desKey);**

**byte[] decryptedBytes = cipher.doFinal(encryptedBytes);**

**String decryptedMessage = new String(decryptedBytes);**

**System.out.println("Decrypted Message : " + decryptedMessage);**

**} catch (NoSuchAlgorithmException | NoSuchPaddingException |**

**InvalidKeyException | IllegalBlockSizeException |**

**BadPaddingException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

1. **Apply AES algorithm for practical applications.**

**import java.security.MessageDigest;**

**import java.security.NoSuchAlgorithmException;**

**import java.util.Arrays;**

**import java.util.Base64;**

**import javax.crypto.Cipher;**

**import javax.crypto.spec.SecretKeySpec;**

**public class AES {**

**private static SecretKeySpec secretKey;**

**private static byte[] key;**

**public static void setKey(String myKey) {**

**try {**

**key = myKey.getBytes("UTF-8");**

**MessageDigest sha = MessageDigest.getInstance("SHA-1");**

**key = sha.digest(key);**

**key = Arrays.copyOf(key, 16); // use only first 128 bits**

**secretKey = new SecretKeySpec(key, "AES");**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**public static String encrypt(String strToEncrypt, String secret) {**

**try {**

**setKey(secret);**

**Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");**

**cipher.init(Cipher.ENCRYPT\_MODE, secretKey);**

**return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));**

**} catch (Exception e) {**

**System.out.println("Error while encrypting: " + e.toString());**

**}**

**return null;**

**}**

**public static String decrypt(String strToDecrypt, String secret) {**

**try {**

**setKey(secret);**

**Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");**

**cipher.init(Cipher.DECRYPT\_MODE, secretKey);**

**return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));**

**} catch (Exception e) {**

**System.out.println("Error while decrypting: " + e.toString());**

**}**

**return null;**

**}**

**public static void main(String[] args) {**

**final String secretKey = "annaUniversity";**

**String originalString = "www.annauniv.edu";**

**String encryptedString = AES.encrypt(originalString, secretKey);**

**String decryptedString = AES.decrypt(encryptedString, secretKey);**

**System.out.println("URL Encryption Using AES Algorithm");**

**System.out.println("------------------------------------------------------------------");**

**System.out.println("Original URL : " + originalString);**

**System.out.println("Encrypted URL : " + encryptedString);**

**System.out.println("Decrypted URL : " + decryptedString);**

**}**

**}**

1. **Implement RSA Algorithm using HTML and JavaScript**

**Html=  
  
<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>RSA Encryption</title>**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<style>**

**body { font-family: Arial; text-align: center; }**

**table { margin: auto; padding: 10px; }**

**td { padding: 5px; }**

**button { padding: 10px 20px; font-size: 16px; }**

**</style>**

**</head>**

**<body>**

**<h1>RSA Algorithm</h1>**

**<h2>Implemented Using HTML & JavaScript</h2>**

**<hr>**

**<table>**

**<tr>**

**<td>Enter First Prime Number (p):</td>**

**<td><input type="number" id="p" value="53"></td>**

**</tr>**

**<tr>**

**<td>Enter Second Prime Number (q):</td>**

**<td><input type="number" id="q" value="59"></td>**

**</tr>**

**<tr>**

**<td>Enter Message (as number):</td>**

**<td><input type="number" id="msg" value="89"></td>**

**</tr>**

**<tr>**

**<td>Public Key (n):</td>**

**<td><span id="publickey"></span></td>**

**</tr>**

**<tr>**

**<td>Exponent (e):</td>**

**<td><span id="exponent"></span></td>**

**</tr>**

**<tr>**

**<td>Private Key (d):</td>**

**<td><span id="privatekey"></span></td>**

**</tr>**

**<tr>**

**<td>Encrypted Message:</td>**

**<td><span id="ciphertext"></span></td>**

**</tr>**

**<tr>**

**<td>Decrypted Message:</td>**

**<td><span id="decryptedtext"></span></td>**

**</tr>**

**<tr>**

**<td colspan="2">**

**<button onclick="RSA()">Apply RSA</button>**

**</td>**

**</tr>**

**</table>**

**<!-- Link to External JavaScript -->**

**<script src="rsa.js"></script>**

**</body>**

**</html>  
  
java++++++++++++++++++++++++++++++++++  
  
function gcd(a, b) {**

**return b === 0 ? a : gcd(b, a % b);**

**}**

**function modPow(base, exponent, modulus) {**

**if (modulus === 1) return 0;**

**let result = 1;**

**base = base % modulus;**

**while (exponent > 0) {**

**if (exponent % 2 === 1)**

**result = (result \* base) % modulus;**

**exponent = Math.floor(exponent / 2);**

**base = (base \* base) % modulus;**

**}**

**return result;**

**}**

**function RSA() {**

**const p = parseInt(document.getElementById('p').value);**

**const q = parseInt(document.getElementById('q').value);**

**const msg = parseInt(document.getElementById('msg').value);**

**const n = p \* q;**

**const phi = (p - 1) \* (q - 1);**

**let e = 2;**

**while (e < phi && gcd(e, phi) !== 1) {**

**e++;**

**}**

**let d;**

**for (let i = 1; ; i++) {**

**let x = 1 + i \* phi;**

**if (x % e === 0) {**

**d = x / e;**

**break;**

**}**

**}**

**const encrypted = modPow(msg, e, n);**

**const decrypted = modPow(encrypted, d, n);**

**document.getElementById('publickey').innerText = n;**

**document.getElementById('exponent').innerText = e;**

**document.getElementById('privatekey').innerText = d;**

**document.getElementById('ciphertext').innerText = encrypted;**

**document.getElementById('decryptedtext').innerText = decrypted;**

**}**

1. **Implement the Diffie-Hellman Key Exchange algorithm for a given problem.**

**public class DiffieHellman {**

**public static void main(String[] args) {**

**int p = 23; // Publicly known prime number**

**int g = 5; // Publicly known primitive root modulo p**

**int a = 4; // Alice's private key**

**int b = 3; // Bob's private key**

**// Alice computes A = g^a mod p and sends it to Bob**

**int A = (int) Math.pow(g, a) % p;**

**// Bob computes B = g^b mod p and sends it to Alice**

**int B = (int) Math.pow(g, b) % p;**

**// Alice computes the shared secret: s = B^a mod p**

**int secretA = (int) Math.pow(B, a) % p;**

**// Bob computes the shared secret: s = A^b mod p**

**int secretB = (int) Math.pow(A, b) % p;**

**System.out.println("Simulation of Diffie-Hellman Key Exchange Algorithm");**

**System.out.println("----------------------------------------------------");**

**System.out.println("Publicly Known Prime (p): " + p);**

**System.out.println("Publicly Known Base (g): " + g);**

**System.out.println("Alice's Private Key (a): " + a);**

**System.out.println("Bob's Private Key (b): " + b);**

**System.out.println("\nAlice Sends (A = g^a % p): " + A);**

**System.out.println("Bob Sends (B = g^b % p): " + B);**

**System.out.println("\nAlice Computes Shared Secret (B^a % p): " + secretA);**

**System.out.println("Bob Computes Shared Secret (A^b % p): " + secretB);**

**if (secretA == secretB) {**

**System.out.println("\n✅ Success: Shared Secrets Match! Shared Secret = " + secretA);**

**} else {**

**System.out.println("\n❌ Error: Shared Secrets Do Not Match!");**

**}**

**}**

**}**

1. **Calculate the message digest of a text using the SHA-1 algorithm.**

**import java.security.MessageDigest;**

**public class SHA1Digest {**

**public static void main(String[] args) {**

**try {**

**MessageDigest md = MessageDigest.getInstance("SHA-1");**

**System.out.println("Message Digest Information");**

**System.out.println("-----------------------------------------------------------");**

**System.out.println("Algorithm : " + md.getAlgorithm());**

**System.out.println("Provider : " + md.getProvider());**

**System.out.println("ToString : " + md.toString());**

**System.out.println();**

**// Test inputs**

**String[] inputs = {**

**"",**

**"abc",**

**"abcdefghijklmnopqrstuvwxyz"**

**};**

**for (String input : inputs) {**

**md.update(input.getBytes());**

**byte[] output = md.digest();**

**System.out.println("SHA1(\"" + input + "\") = " + bytesToHex(output));**

**System.out.println();**

**}**

**} catch (Exception e) {**

**System.out.println("Exception: " + e.getMessage());**

**}**

**}**

**private static String bytesToHex(byte[] bytes) {**

**StringBuilder hexString = new StringBuilder();**

**for (byte b : bytes) {**

**String hex = Integer.toHexString(0xff & b).toUpperCase();**

**if (hex.length() == 1) hexString.append('0');**

**hexString.append(hex);**

**}**

**return hexString.toString();**

**}**

**}**

1. **Implement the Signature Scheme - Digital Signature Standard.**

**import java.security.KeyPair;**

**import java.security.KeyPairGenerator;**

**import java.security.PrivateKey;**

**import java.security.PublicKey;**

**import java.security.Signature;**

**import java.util.Scanner;**

**import java.util.Base64;**

**public class CreatingDigitalSignature {**

**public static void main(String[] args) throws Exception {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("Enter some text to sign:");**

**String msg = sc.nextLine();**

**sc.close();**

**// Step 1: Generate Key Pair**

**KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");**

**keyPairGen.initialize(2048); // key size**

**KeyPair pair = keyPairGen.generateKeyPair();**

**// Step 2: Create a signature object**

**Signature sign = Signature.getInstance("SHA256withDSA");**

**// Step 3: Sign the message**

**PrivateKey privKey = pair.getPrivate();**

**sign.initSign(privKey);**

**sign.update(msg.getBytes());**

**byte[] signatureBytes = sign.sign();**

**// Step 4: Print the digital signature in base64 (readable form)**

**String signatureBase64 = Base64.getEncoder().encodeToString(signatureBytes);**

**System.out.println("\nDigital Signature (Base64 Encoded):");**

**System.out.println(signatureBase64);**

**// Step 5: Verifying the signature (optional for completeness)**

**Signature verifySign = Signature.getInstance("SHA256withDSA");**

**PublicKey pubKey = pair.getPublic();**

**verifySign.initVerify(pubKey);**

**verifySign.update(msg.getBytes());**

**boolean isVerified = verifySign.verify(signatureBytes);**

**System.out.println("\nSignature Verification Result: " + (isVerified ? "Valid ✅" : "Invalid ❌"));**

**}**

**}**

1. **Demonstrate intrusion detection system (IDS) using any tool ex. Snort or any other s/w.**
2. **Automated Attack and Penetration Tools Exploring N-Stalker, a Vulnerability Assessment Tool.**