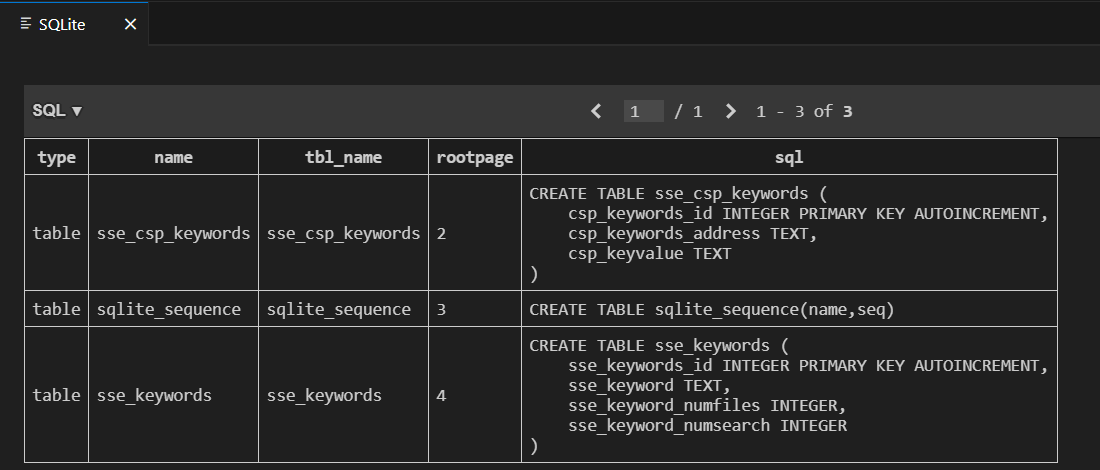
Tutorial 8

I am using sqlite3 with python to set up my tables

Screen shot of both tables



A screenshot of a computer

Description automatically generated

Set up script

A screen shot of a computer program

Description automatically generated

Rest of the work by the script

import sqlite3

import os

import hashlib

import time

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default\_backend

import secrets

from cryptography.hazmat.primitives import padding

# Connect to the SQLite database

conn = sqlite3.connect('./sse\_schema.db')

cursor = conn.cursor()

# Create necessary tables if they don't exist

cursor.execute('''CREATE TABLE IF NOT EXISTS sse\_csp\_keywords (

    csp\_keywords\_id INTEGER PRIMARY KEY AUTOINCREMENT,

    csp\_keywords\_address TEXT,

    csp\_keyvalue TEXT

)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS sse\_keywords (

    sse\_keywords\_id INTEGER PRIMARY KEY AUTOINCREMENT,

    sse\_keyword TEXT,

    sse\_keyword\_numfiles INTEGER,

    sse\_keyword\_numsearch INTEGER

)''')

conn.commit()

# Define folder path for text files

folder\_path = './textfiles/'

# Define AES key (use your actual KSKE here)

KSKE\_hex = '8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91'

KSKE = bytes.fromhex(KSKE\_hex)

# AES encryption function with padding

def aes\_encrypt(data, key):

    iv = secrets.token\_bytes(16)

    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default\_backend())

    encryptor = cipher.encryptor()

    # Pad data to ensure it's a multiple of 16 bytes (AES block size)

    padder = padding.PKCS7(algorithms.AES.block\_size).padder()

    padded\_data = padder.update(data) + padder.finalize()

    return iv + encryptor.update(padded\_data) + encryptor.finalize()

# SHA-256 hash function for keywords

def hash\_word(word):

    return hashlib.sha256(word.encode()).hexdigest()

# Keyword key computation

def keyword\_key(keyword, numsearch):

    combined = f"{keyword}{numsearch}".encode()

    return hashlib.sha256(combined).hexdigest()

# Compute CSP keywords address

def csp\_keyword\_address(Kw, numfiles):

    combined = f"{Kw}{numfiles}".encode()

    return hashlib.sha256(combined).hexdigest()

# Insert or update keyword in the database

def insert\_or\_update\_keyword(cursor, keyword\_hash, numfiles, numsearch):

    # Check if the keyword already exists

    cursor.execute('''

    SELECT sse\_keyword\_numfiles, sse\_keyword\_numsearch FROM sse\_keywords WHERE sse\_keyword = ?

    ''', (keyword\_hash,))

    result = cursor.fetchone()

    if result:

        # Keyword exists, update numfiles and numsearch

        existing\_numfiles, existing\_numsearch = result

        new\_numfiles = existing\_numfiles + numfiles  # Increment numfiles

        # new\_numsearch = existing\_numsearch + 1       # Increment numsearch

        cursor.execute('''

        UPDATE sse\_keywords SET sse\_keyword\_numfiles = ?

        WHERE sse\_keyword = ?

        ''', (new\_numfiles, keyword\_hash))

    else:

        # Keyword doesn't exist, insert new record

        cursor.execute('''

        INSERT INTO sse\_keywords (sse\_keyword, sse\_keyword\_numfiles, sse\_keyword\_numsearch)

        VALUES (?, ?, ?)

        ''', (keyword\_hash, numfiles, numsearch))

# Collect all CSP keywords data for later processing

def collect\_csp\_keywords\_data(cursor):

    cursor.execute('''

    SELECT sse\_keyword, sse\_keyword\_numfiles FROM sse\_keywords

    ''')

    return cursor.fetchall()

total\_size = sum(os.path.getsize(os.path.join(folder\_path, f)) for f in os.listdir(folder\_path) if f.endswith('.txt'))

# Start execution timer

start\_time = time.time()

# First Pass: Process each file to update the sse\_keywords table

for filename in os.listdir(folder\_path):

    if filename.endswith('.txt'):

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

        # Read the original file to extract keywords

        with open(file\_path, 'r') as file:

            words = file.read().split()

        numfiles = 1  # Each file contains these keywords at least once

        # Process each word to update the sse\_keywords table

        for word in words:

            keyword\_hash = hash\_word(word)

            insert\_or\_update\_keyword(cursor, keyword\_hash, numfiles, 1)

# Second Pass: Encrypt and delete all files

for filename in os.listdir(folder\_path):

    if filename.endswith('.txt'):

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

        # Read and encrypt the file

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

            data = file.read()

            encrypted\_data = aes\_encrypt(data, KSKE)

        # Save encrypted file and delete the original

        with open(file\_path + '.enc', 'wb') as encrypted\_file:

            encrypted\_file.write(encrypted\_data)

        os.remove(file\_path)  # Delete original .txt file

# Third Pass: Compute and insert CSP keywords based on the populated sse\_keywords table

csp\_keywords\_data = []

for keyword, numfiles in collect\_csp\_keywords\_data(cursor):

    kw = keyword\_key(keyword, 1)

    # Print parameters before calculating CSP address

    # print(f"Calculating CSP address with keyword: '{kw}' and numfiles: {numfiles}")

    address = csp\_keyword\_address(kw, numfiles)

    encrypted\_filename = aes\_encrypt(f"{filename}{numfiles}".encode(), KSKE)

    # print(f"Encrypted with aes: {filename} + {numfiles}")

    csp\_keywords\_data.append((address, encrypted\_filename))

# Insert collected CSP keywords data into the database

for address, encrypted\_filename in csp\_keywords\_data:

    cursor.execute('''

    INSERT INTO sse\_csp\_keywords (csp\_keywords\_address, csp\_keyvalue)

    VALUES (?, ?)

    ''', (address, encrypted\_filename))

# Commit all changes

conn.commit()

# End execution timer

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"Total execution time: {execution\_time:.2f} seconds")

print(f"Total combined size of test files: {total\_size} bytes")

conn.close()

I generate 25 random text files as

import os

import random

# Folder path for the generated .txt files

folder\_path = 'textfiles/'

# Ensure the folder exists

os.makedirs(folder\_path, exist\_ok=True)

# List of common English words

word\_list = [

    "apple", "banana", "orange", "grape", "pineapple", "strawberry", "blueberry", "mango", "peach", "lemon",

    "cherry", "watermelon", "pear", "plum", "kiwi", "melon", "lime", "coconut", "apricot", "grapefruit",

    "berry", "papaya", "guava", "fig", "date", "pomegranate", "citrus", "nectarine", "raspberry", "blackberry"

]

# Function to generate random text content with one unique word per line

def generate\_random\_text():

    # Choose a random number of unique words for the file (between 5 and 30)

    num\_words = random.randint(5, min(30, len(word\_list)))  # Ensure it doesn't exceed the number of unique words

    chosen\_words = random.sample(word\_list, k=num\_words)  # Choose unique words

    return '\n'.join(chosen\_words)  # Join words with line breaks

# Generate 25 .txt files with varied sizes

for i in range(1, 26):

    file\_content = generate\_random\_text()  # Generate unique words for each file

    file\_path = os.path.join(folder\_path, f'test\_file\_{i}.txt')

    with open(file\_path, 'w') as f:

        f.write(file\_content)

print("25 .txt files with random unique words per line have been created.")

It would be much better if 25 txt files are provided since I had to changed this code many times to get possible files.

Output of the script

aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial8 (main)$ python3 database.py

KSKE is : 8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91

Total execution time: 8.82 seconds

Total combined size of test files: 2909 bytes

Searching script

import hashlib

import time

import sqlite3

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default\_backend

import re

# Define your AES key here (the KSKE from the previous script)

KSKE = '8da5ed8fdd57592c388e638fb56b82fecf3e563c37897a97893b33b4308f9c91'  # Replace this with your actual KSKE key from earlier

KSKE = bytes.fromhex(KSKE)

# AES decryption function

def aes\_decrypt(data, key):

    iv = data[:16]

    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default\_backend())

    decryptor = cipher.decryptor()

    return decryptor.update(data[16:]) + decryptor.finalize()

# SHA-256 hash function for keywords

def hash\_word(word):

    print(f"Hash of {word} is: ", hashlib.sha256(word.encode()).hexdigest())

    return hashlib.sha256(word.encode()).hexdigest()

# Keyword key computation

def keyword\_key(keyword, numsearch):

    combined = f"{keyword}{numsearch}".encode()

    return hashlib.sha256(combined).hexdigest()

# Compute csp\_keywords\_address

def csp\_keyword\_address(Kw, numfiles):

    combined = f"{Kw}{numfiles}".encode()

    return hashlib.sha256(combined).hexdigest()

# Connect to the database

conn = sqlite3.connect('./sse\_schema.db')

cursor = conn.cursor()

# Start search timer

start\_time = time.time()

# 1. Ask Bob to enter a word to search for

search\_word = input("Enter the word you want to search for: ")

# 2. Create SHA-256 hash of the word to generate the keyword value

keyword\_hash = hash\_word(search\_word)

# 3. Retrieve numfiles and numsearch for the keyword from sse\_keywords table

cursor.execute('''

SELECT sse\_keyword\_numfiles, sse\_keyword\_numsearch FROM sse\_keywords WHERE sse\_keyword = ?

''', (keyword\_hash,))

result = cursor.fetchone()

if result:

    numfiles, numsearch = result

    # 4. Compute the keyword key Kw

    Kw = keyword\_key(keyword\_hash, numsearch)

    # print(Kw, numsearch)

    # 5. Compute csp\_keywords\_address

    csp\_address = csp\_keyword\_address(Kw, numfiles)

    # print(csp\_address, numfiles)

    # 6. Retrieve csp\_keyvalue using csp\_keywords\_address

    cursor.execute('''

    SELECT csp\_keyvalue FROM sse\_csp\_keywords WHERE csp\_keywords\_address = ?

    ''', (csp\_address,))

    encrypted\_filename\_result = cursor.fetchall()

    if encrypted\_filename\_result:

        # If results are found, decrypt each associated file

        for (encrypted\_filename,) in encrypted\_filename\_result:

            print(encrypted\_filename.hex())

            # 7. Decrypt the filename and numfiles from csp\_keyvalue

            decrypted\_info = aes\_decrypt(encrypted\_filename, KSKE).decode()

            print("Checking: ", decrypted\_info)

            decrypted\_info = re.sub(r'\\.\*$', '', repr(decrypted\_info))

            match = re.match(r"(.+?)(\d+)$", decrypted\_info)

            if match:

                filename = match.group(1).lstrip("'")   # Extracts 'test\_file\_1.txt'

                file\_num = match.group(2)   # Extracts '1'

                print("Filename:", filename)

                print("File Number:", file\_num)

                # 8. Decrypt the file content

                file\_path = f'./textfiles/{filename}.enc'

                try:

                    with open(file\_path, 'rb') as encrypted\_file:

                        encrypted\_data = encrypted\_file.read()

                    decrypted\_data = aes\_decrypt(encrypted\_data, KSKE).decode()

                    print(f"Decrypted content of {filename}:")

                    print(decrypted\_data)

                except FileNotFoundError:

                    print(f"File {file\_path} not found.")

            else:

                print("Format error in decrypted\_info.")

    else:

        print("No CSP key values found for the specified keyword.")

else:

    print("No files found containing the specified word.")

# Measure end time and calculate search time

end\_time = time.time()

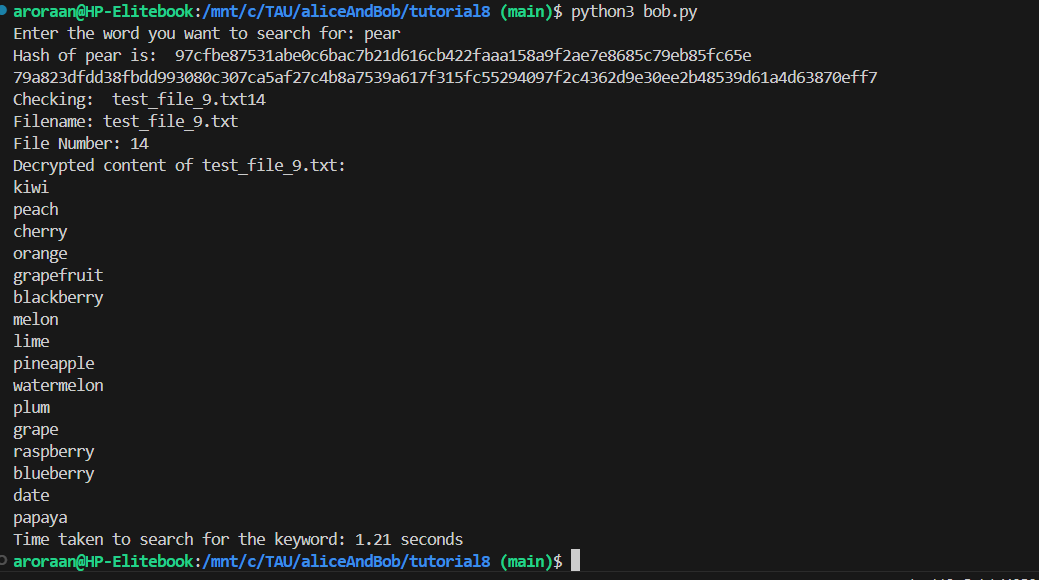
search\_time = end\_time - start\_time

print(f"Time taken to search for the keyword: {search\_time:.2f} seconds")

# Close database connection

conn.close()

output



If word not found

A screenshot of a computer program

Description automatically generated

This assignment would be way easier if txt files were given and csp\_keyword updating and when to update numfiles and numsearch was given in more detail. Currently the tutorial takes focus away from actual working of database due to these other issues.