CYBER SECURITY & ETHICAL HACKING

Group-5

Instructor:
Abdur Rehman

PROJECTS:

- MANAGING EMAIL SPOOFING THROUGH EMAIL ADDRESS ANALYSIS USING MXTOOLBOX.
- FILE ENCRYPTION AND INTEGRITY MANAGEMENT USING PYTHON AID SQ-LITE

GROUP MEMBERS:

SHEEZA ALAM KHAN
HOORIYA EHTESHAM
HAREEM MALLICK

PROJECT#01

Email Spoof Detection and Verification through Header Analysis

---

Objective

The main objective of this project is to identify and verify whether a received email is genuine or spoofed. This is achieved by performing an in-depth analysis of the email header using the MX Toolbox Email Header Analyzer. The project demonstrates how header details, authentication checks such as SPF, DKIM, and DMARC, and relay information can reveal the true origin and authenticity of an email message.

Tools and Environment

Tool Used: MXToolbox – https://mxtoolbox.com/EmailHeaders.aspx
Operating System: Kali Linux / Windows
Browser: Google Chrome or Firefox
Email Source: Gmail message header (from ACCA Careers)

Introduction

Email spoofing has become a common method used by attackers to trick recipients into believing that a fraudulent email is from a trusted source. This project aims to understand how email headers and authentication mechanisms can be analyzed to verify the authenticity of an email. By studying the raw header information and using MXToolbox for structured analysis, it is possible to determine whether an email originated from a legitimate source or has been spoofed.

Overview

An email header contains technical details about how an email was transmitted from the sender to the receiver. It includes routing paths, IP addresses, timestamps, and authentication checks that help trace the message's journey. Analyzing these details allows cybersecurity professionals to verify whether the email genuinely comes from the claimed domain or if it was forged.

Modern email systems use authentication standards such as SPF, DKIM, and DMARC to ensure sender legitimacy. By analyzing these records, one can confirm whether the email server is authorized to send messages for a given domain and whether the message has remained unaltered during transmission.

Motivation

Email spoofing is widely used by cybercriminals to perform phishing attacks, deliver malware, or steal sensitive information. To counter these threats, authentication mechanisms have been developed to validate the sender's identity and ensure message integrity.

SPF (Sender Policy Framework) verifies if the mail server is authorized to send emails for the domain.
DKIM (DomainKeys Identified Mail) uses cryptographic signatures to confirm that the message has not been modified during transit.
DMARC (Domain-based Message Authentication, Reporting, and Conformance) combines SPF and DKIM to decide whether an email should be accepted, quarantined, or rejected.

Understanding and analyzing these authentication methods through email headers allows us to distinguish legitimate communications from spoofed or malicious ones.

Methodologies

Step 1: Obtaining the Email Header
The suspicious or target email was opened in Gmail, and the "Show Original" option was used to access the full email header. The entire header text was then copied for analysis.

Step 2: Using MXToolbox
The copied header was pasted into the MXToolbox Email Header Analyzer, which processes and presents detailed information about authentication results, relay paths, and delivery times. This tool identifies SPF, DKIM, and DMARC outcomes, as well as the mail servers that handled the message.

End Result

The analyzed email header belonged to a confirmation message from ACCA Careers with the subject *"Application submitted confirmation (ACCA Careers)"*.

MXToolbox produced the following results:

- **SPF Alignment:** OK – The sending domain matched correctly.
- **SPF Authentication:** Minor lookup issue observed but no direct failure.
- **DKIM Alignment:** OK – The digital signature aligned properly.
- **DMARC Compliance:** Partial failure noted, but overall authentication passed successfully.

The relay path showed that the email traveled from *mta126.madgexjb.com* (IP: 52.209.63.219) to *mx.google.com*, confirming that the sender used Amazon AWS infrastructure via Amazon Route 53 for email delivery. The one-second delivery delay indicated a direct and legitimate transmission.

SPF verification confirmed that the IP address 52.209.63.219 is authorized to send emails for the domain *accaglobal.com*. DKIM used a strong RSA-SHA256 cryptographic signature, and Gmail's authentication confirmed that DKIM, SPF, and DMARC all passed.

The *Return-Path* and *From* address were identical (*accacareers@accaglobal.com*), showing that the sender's identity was not forged, which supports the conclusion that the email was authentic.

What You Have Learned

Through this project, the following key learnings were achieved:

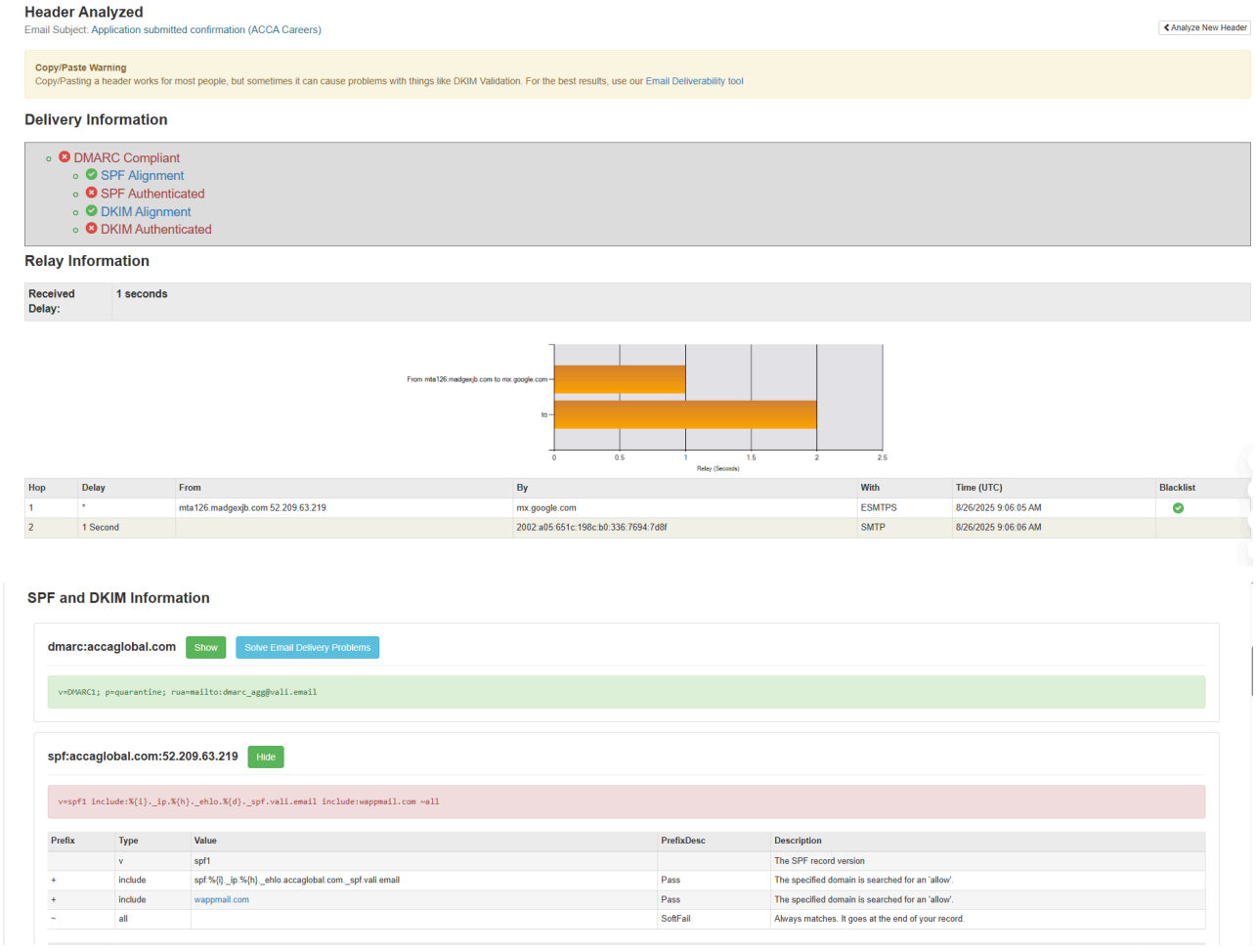
1. How to extract and interpret raw email headers to trace message origins.
2. The working principles and importance of SPF, DKIM, and DMARC in preventing spoofed emails.
3. How to effectively use MXToolbox for header analysis and authentication verification.
4. Methods to differentiate between legitimate and suspicious emails using technical data.

This process enhanced understanding of real-world email security mechanisms and how analytical tools can aid in protecting against phishing and spoofing attacks.

Conclusion

The email from ACCA Careers successfully passed SPF, DKIM, and DMARC verification checks at Gmail's servers. The message originated from an authorized Amazon AWS IP and followed a legitimate relay path. Although MXToolbox reported minor intermediate issues, the final authentication confirmed that the email was genuine and not spoofed.

This project demonstrates how analyzing email headers and authentication mechanisms provides a practical and effective method to detect spoofing attempts, ensuring better email security and user trust.



Header Analyzed

Email Subject: Application submitted confirmation (ACCA Careers)

Analyze New Header

Copy/Paste Warning

Copy/Pasting a header works for most people, but sometimes it can cause problems with things like DKIM Validation. For the best results, use our Email Deliverability tool

Delivery Information

- DMARC Compliant
  - SPF Alignment
  - SPF Authenticated
  - DKIM Alignment
  - DKIM Authenticated

Relay Information

Received Delay:	1 seconds

Hop	Delay	From	By	With	Time (UTC)	Blacklist
1	*	mta126.madgexjb.com 52.209.63.219	mx.google.com	ESMTPS	8/26/2025 9:06:05 AM	
2	1 Second		2002:a05:651c:198c:b0:336:7694:7d8f	SMTP	8/26/2025 9:06:06 AM	

SPF and DKIM Information

dmarc:accaglobal.com Show Solve Email Delivery Problems

```
v=DMARC1; p=quarantine; rua=mailto:dmarc_agg@vali.email
```

spf:accaglobal.com:52.209.63.219 Hide

```
v=spf1 include:%{i}._ip.%{h}._ehlo.%{d}._spf.vali.email include:wappmail.com ~all
```

Prefix	Type	Value	PrefixDesc	Description
	v	spf1		The SPF record version
+	include	spf.%{i} _ip.%{h} _ehlo.accaglobal.com _spf.vali.email	Pass	The specified domain is searched for an 'allow'.
+	include	wappmail.com	Pass	The specified domain is searched for an 'allow'.
~	all		SoftFail	Always matches. It goes at the end of your record.

	Test	Result
✖	SPF Authentication	SPF Failed for IP - 52.209.63.219
✔	SPF Record Published	SPF Record found
✔	SPF Record Deprecated	No deprecated records found
✔	SPF Multiple Records	Less than two records found
✔	SPF Alignment	Domain found in SPF
✔	SPF Contains characters after ALL	No items after 'ALL'.
✔	SPF Syntax Check	The record is valid
✔	SPF Included Lookups	Number of included lookups is OK
✔	SPF Recursive Loop	Nor Recursive Loops on Includes
✔	SPF Duplicate Include	No Duplicate Includes Found
✔	SPF Type PTR Check	No type PTR found
✔	SPF Void Lookups	Number of void lookups is OK
✔	SPF MX Resource Records	Number of MX Resource Records is OK
✔	SPF Record Null Value	No Null DNS Lookups found

Your DNS hosting provider is "Amazon Route 53" Need Bulk Dns Provider Data?

dkim:accaglobal.com:madgex Show

Dkim Public Record:

k=rsa; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDIcnh+CB7dO2eeoleE72DRlyXZc6LRzmgYCeD0Gh4Ekb8pIkPj/7hAPIGULrroO3dgNj/Z3AuveuBp0aN/j5gOBSyJr62+y2sRja3sZyuvSERhvgRy677F0hxGuxiYbY6VqDi5qcYj8LOfkVdII75G15ncOYBliH6CN2o33tBLwIDAQAB

Dkim Signature:

v=1; a=rsa-sha256; c=relaxed; s=madgex; d=accaglobal.com; h=Message-ID:MIME-Version:From:To:Reply-To:Date:Subject:Content-Type; i=accacareers@accaglobal.com; bh=UrgS5toHzq9BsAf84XnjnVw+YYEaXZwtVyPwLL/a05A=; b=LcAirqsRHsLXgDyP01nxpFz5+zILNSRckr8

Headers Found

Header Name	Header Value
Delivered-To	shizaalam50@gmail.com
X-Google-Smtp-Source	AGHT+IEfz2iQM4BbhDL/F6Q4z/CLIVDxVAlFOYvOFZQLaG71CbknvbjLW4IK+IUHtgfHbY10jSN/
X-Received	by 2002:a05:6000:2403:b0:3c9:1433:f8c0 with SMTP id ffacd0b85a97d-3c91433ff3fmr5804897f8f.7.1756199165918; Tue, 26 Aug 2025 02:06:05 -0700 (PDT)
ARC-Seal	i=1; a=rsa-sha256; t=1756199165; cv=none; d=google.com; s=arc-20240605; b=KQVdGuLUhn3p4b6C22bELwSmjh7B9/47hwp0n1S/4n7FAOwx+TI8nUKgmWCqd+wrlj yW+S2udwlTJvL+GT62uqK4ca0Vc3aPJLKsqbmfqldYfX4MR/lz+1677c2fWNYMgX/QHD Re/h78hexps+8PeXh4Vy+c592wSMQVUBIOFdYq4k1zRmSTnGm257P8dDd9s76YHgdBQX qX8QpQJA8/SVdQHbW4vP3JYYcZ/fNlwu2/reO6JdjJ7Xvh5ntad7uNQWbrdy6pz4YzKK tMFKsmenFSO7V4lM4qv0uTi69cEcDIUUA5phbgnv3NLxBCgKfh95ZINFAddw8G7AnkvU fLHg==
ARC-Message-Signature	i=1; a=rsa-sha256; c=relaxed/relaxed; d=google.com; s=arc-20240605; h=subject:date:reply-to:to:from:mime-version:message-id :dkim-signature; bh=UrgS5toHzq9BsAf84XnjnVw+YYEaXZwtVyPwLL/a05A=; fh=Ksc/L8Be8YjAkc2NeLNkFqNNFCElebOV6xls535pM8c=; b=Qqh6/kN03EjLdl s4aTJy+6Kw/wMWHG1Jf0XPuHBTRR0o3TXV5ansypf9uwz9XLC6Di N+n594yskYLBTwP2oHkQ+OeAj/IsNz4lOWVuk1N2ED/5ijLiIEbHnUWeTuc30iFMZn17 4E95bmuVDMEsCFetuLR9ZqYxqMZ90/XyGR2RAG2qxUXgpj5Ow3JSgYan8+nmhOKvvbOP 2iCEzVOAypX3UuBva5crMgyZQk1ZI c6ynag5HWw6QaOfTqWkQq2AX7IlOBIu2n0A8bgH lzccV0ue/mjD6BITir+n4l4/aZnsOxRlt6sDjXfi+FHUjj9DZqLS+AjDcACR7wC5/A/n raGg==; dara=google.com
ARC-Authentication-Results	i=1; mx.google.com; dkim=pass header.i=@accaglobal.com header.s=madgex header.b=LcAirqsR; spf=pass (google.com: domain of accacareers@accaglobal.com designates 52.209.63.219 as permitted sender) smtp.mailfrom=accacareers@accaglobal.com; dmarc=pass (p=QUARANTINE sp=QUARANTINE dis=NONE) header.from=accaglobal.com
Return-Path	<accacareers@accaglobal.com>
Received-SPF	pass (google.com: domain of accacareers@accaglobal.com designates 52.209.63.219 as permitted sender) client-ip=52.209.63.219;
Authentication-Results	mx.google.com; dkim=pass header.i=@accaglobal.com header.s=madgex header.b=LcAirqsR; spf=pass (google.com: domain of accacareers@accaglobal.com designates 52.209.63.219 as permitted sender) smtp.mailfrom=accacareers@accaglobal.com; dmarc=pass (p=QUARANTINE sp=QUARANTINE dis=NONE) header.from=accaglobal.com
DKIM-Signature	v=1; a=rsa-sha256; c=relaxed; s=madgex; d=accaglobal.com; h=Message-ID:MIME-Version:From:To:Reply-To:Date:Subject:Content-Type; i=accacareers@accaglobal.com; bh=UrgS5toHzq9BsAf84XnjnVw+YYEaXZwtVyPwLL/a05A=; b=LcAirqsRHsLXgDyP01nxpFz5+zILNSRckr8+TGhxk GAZjXzH5riWq+JiyileLrl1qgzvlRm+druH mX7ETa/7UZGR0f4fvURnnDiuGT5hsBon1Hq5ZW+shx1VSlrNhPw/ynOPSDK3H9q4lmrKtWhd3obO t4jiWjk5jGzoEB9ZojU=
Message-ID	<M_592e60df-adbc-4a6d-a5cc-9dac5275ed22@ACCACareers>
X-Madgex-SystemEmailName	ActivationFromApplication
X-Madgex-SystemEmailType	UserGeneratedSystemEmail
MIME-Version	1.0
From	"ACCA Careers (no reply)" <accacareers@accaglobal.com>
To	shizaalam50@gmail.com
Reply-To	"ACCA Careers (no reply)" <accacareers@accaglobal.com>
Date	26 Aug 2025 10:06:05 +0100
Subject	Application submitted confirmation (ACCA Careers)
Content-Type	multipart/mixed; boundary=--boundary_870_2c521542-e8ff-46f7-a464-e21aaa43f553
X-OriginalArrivalTime	26 Aug 2025 09:06:05.0230 (UTC) FILETIME=[A73A54E0:01DC1668]

PROJECT #02

File Encryption and Integrity Management using Python and SQLite

Introduction

This project focuses on developing a Python-based tool to ensure file confidentiality and integrity. It performs secure encryption, decryption, and file integrity verification using cryptographic techniques. The system stores hash values in an SQLite database for later verification, ensuring that any unauthorized modification of files can be detected effectively.

Overview

The project implements AES-256 encryption for data confidentiality and SHA-256 hashing for integrity verification. It allows users to encrypt, decrypt, and verify files through a command-line interface. All operations are backed by a local SQLite database that securely stores file hashes for authenticity checks.

Objective

The objective of this project is to design and implement a secure, efficient, and user-friendly tool that:

- Encrypts and decrypts files using AES-256 encryption.
- Verifies file integrity using SHA-256 hash values.
- Stores integrity information in an SQLite database for future verification.

Tools and Technologies

Programming Language: Python 3
Database: SQLite
Operating System: Kali Linux
File Path: `/home/shiza/Desktop/filetool.py`

Libraries Used:

- `hashlib` – For generating SHA-256 file hashes.
- `sqlite3` – For database creation and management.
- `argparse` – For command-line interface support.
- `pycryptodome` – For AES-256 encryption and decryption.

Motivation

In the digital era, data confidentiality and integrity are essential to prevent unauthorized access and tampering. This project combines cryptography and integrity management to secure sensitive files. By integrating AES encryption with SHA-256 hashing, it ensures both data protection and verification of file authenticity.

Methodologies

1. **Database Initialization**
    - A database named `file_integrity.db` is created to store filenames and their SHA-256 hashes.
2. **File Hashing**

SHA-256 algorithm generates a unique digital fingerprint for each file using:

hashlib.sha256(file_contents).hexdigest()

    -
3. **File Encryption**

- Files are encrypted using AES-256 in GCM mode with password-based key derivation (PBKDF2).
- A tag ensures data authenticity and prevents unauthorized decryption.

4. **File Decryption**
   - Files are decrypted using the same password.
   - If authentication fails, an alert indicates tampering or an incorrect password.
5. **Integrity Storage and Verification**
   - The computed hash is stored in SQLite.
   - The tool compares the current hash with the stored one to detect modifications.
6. **Demo Function**
   - A demo automatically performs all steps:
     - Creates `secret.txt`
     - Encrypts → `secret.txt.enc`
     - Stores hash in the database
     - Verifies integrity
     - Decrypts → `secret_decrypted.txt`

Execution Programs

To Run the Project:

cd /home/shiza/Desktop
python3 filetool.py demo

Manual Commands:

Encrypt a file:

python3 filetool.py encrypt secret.txt -p "MyPass123" -o secret.enc

-

Decrypt a file:

python3 filetool.py decrypt secret.enc -p "MyPass123" -o secret_decrypted.txt

-

Store file hash:

python3 filetool.py integrity-check secret.txt --store

-

Verify file integrity:

python3 filetool.py integrity-check secret.txt

RESULTS

Function	Output
Encryption	Displays: “File encrypted successfully → secret.enc”
Decryption	Displays: “File decrypted successfully → secret_decrypted.txt”
Integrity Check (Match)	“File integrity OK — file has not been tampered with.”
Integrity Check (Mismatch)	“ALERT: file has been modified or tampered!”

Error Handling

- **Incorrect Password:** Displays “Decryption failed! The password may be incorrect or the file was tampered with.”
- **Missing File or Database:** Displays informative warnings to reinitialize or store the hash again.
- Ensures smooth execution with meaningful, user-friendly messages.

What You Have Learned

- Implementing AES-256 encryption and decryption in Python.
- Using SHA-256 hashing for digital fingerprinting.
- Managing file integrity data using SQLite databases.
- Designing secure command-line tools with error handling.
- Detecting and preventing unauthorized file tampering.

End Result

The developed tool provides a secure system for encrypting, decrypting, and verifying files. It ensures confidentiality through AES encryption, maintains integrity through SHA-256 hashing, and uses SQLite for accountability and record-keeping.

Conclusion

The project successfully demonstrates a comprehensive File Encryption and Integrity Management System using Python and SQLite. It ensures data confidentiality, integrity, and accountability, providing an effective solution for cybersecurity applications. This tool serves as a reliable asset for both students and professionals to safeguard sensitive information and detect unauthorized file modifications.

```
┌──(shiza㉿kali)-[~]
└─$ python3 -m venv venv
source venv/bin/activate
pip install pycryptodome
Requirement already satisfied: pycryptodome in ./venv/lib/python3.13/site-pac
kages (3.23.0)

┌──(venv)─(shiza㉿kali)-[~]
└─$ cd Desktop

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py demo

── Demo: Encrypting File ──
🔒 File encrypted successfully → /home/shiza/Desktop/secret.txt.enc

── Demo: Storing Integrity Hash ──
📝 Hash stored in database for integrity check.

── Demo: Checking Integrity ──
✔ File integrity OK — /home/shiza/Desktop/secret.txt has not been tampered w
ith.

── Demo: Decrypting File ──
🔓 File decrypted successfully → /home/shiza/Desktop/secret_decrypted.txt

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py encrypt secret.txt -p "MyPass123" -o secret.enc
🔒 File encrypted successfully → secret.enc

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py decrypt secret.enc -p "MyPass123" -o secret_decrypted
.txt
🔓 File decrypted successfully → secret_decrypted.txt

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py integrity-check secret.txt --store
✔ Stored hash for secret.txt

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py integrity-check secret.txt
✔ File integrity OK — secret.txt has not been tampered with.

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ mousepad secret.txt
```

```
┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ mousepad secret.txt

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ python3 filetool.py integrity-check secret.txt
✗  ALERT: secret.txt has been modified or tampered!

┌──(venv)─(shiza㉿kali)-[~/Desktop]
└─$ cat secret.txt
This is a secret message. Protect it carefully!
hello whatsup??
```

~/Desktop/filetool.py - Mousepad

File Edit Search View Document Help

```python
import os
import sys
import sqlite3
import hashlib
import base64
from Crypto.Cipher import AES
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Random import get_random_bytes
import argparse

# ———————— Database setup ————————
DB_PATH = "/home/shiza/Desktop/file_integrity.db"

def init_db():
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS files (
                    filename TEXT PRIMARY KEY,
                    hash TEXT NOT NULL
                )''')
    conn.commit()
    conn.close()

# ———————— Hashing ————————
def compute_hash(file_path):
    sha256_hash = hashlib.sha256()
    with open(file_path, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            sha256_hash.update(chunk)
    return sha256_hash.hexdigest()

# ———————— Integrity Functions ————————
def store_hash(filename, hash_value):
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("REPLACE INTO files (filename, hash) VALUES (?, ?)", (filename, hash_value))
    conn.commit()
    conn.close()

```

```python
def check_integrity(filename):
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT hash FROM files WHERE filename=?", (filename,))
    row = c.fetchone()
    conn.close()

    if not row:
        print(f"⚠️ No stored hash found for {filename}. Please store it first using integrity-check.")
        return

    stored_hash = row[0]
    current_hash = compute_hash(filename)

    if stored_hash == current_hash:
        print(f"✅ File integrity OK — {filename} has not been tampered with.")
    else:
        print(f"❌ ALERT: {filename} has been modified or tampered!")

# ———— Encryption ————
def encrypt_file(file_path, password, output_path):
    salt = get_random_bytes(16)
    key = PBKDF2(password, salt, dkLen=32)
    cipher = AES.new(key, AES.MODE_GCM)
    with open(file_path, "rb") as f:
        plaintext = f.read()
    ciphertext, tag = cipher.encrypt_and_digest(plaintext)
    with open(output_path, "wb") as f:
        [f.write(x) for x in (salt, cipher.nonce, tag, ciphertext)]
    print(f"🔒 File encrypted successfully → {output_path}")
```

```python
# ———— Decryption ————
def decrypt_file(file_path, password, output_path):
    try:
        with open(file_path, "rb") as f:
            salt, nonce, tag, ciphertext = [f.read(x) for x in (16, 16, 16, -1)]
        key = PBKDF2(password, salt, dkLen=32)
        cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)
        plaintext = cipher.decrypt_and_verify(ciphertext, tag)
        with open(output_path, "wb") as f:
            f.write(plaintext)
        print(f"🔓 File decrypted successfully → {output_path}")
    except ValueError:
        print("❌ Decryption failed! The password may be incorrect or the file was tampered with.")
    except Exception as e:
        print(f"⚠️ Error: {e}")

```

```python
# ———— Demo ————
def demo():
    sample_path = "/home/shiza/Desktop/secret.txt"
    encrypted_path = "/home/shiza/Desktop/secret.txt.enc"
    decrypted_path = "/home/shiza/Desktop/secret_decrypted.txt"
    password = "TestPass123"

    # Create a sample file
    with open(sample_path, "w") as f:
        f.write("This is a secret message. Protect it carefully!")

    print("\n—— Demo: Encrypting File ——")
    encrypt_file(sample_path, password, encrypted_path)

    print("\n—— Demo: Storing Integrity Hash ——")
    file_hash = compute_hash(sample_path)
    store_hash(sample_path, file_hash)
    print(f"📝 Hash stored in database for integrity check.\n")

    print("—— Demo: Checking Integrity ——")
    check_integrity(sample_path)

    print("\n—— Demo: Decrypting File ——")
    decrypt_file(encrypted_path, password, decrypted_path)

```

```python
# ———— Main CLI ————
def main():
    init_db()

    parser = argparse.ArgumentParser(description="File Encryption + Integrity Checker Tool")
    subparsers = parser.add_subparsers(dest="command")

    # Encrypt
    enc = subparsers.add_parser("encrypt")
    enc.add_argument("file", help="Path to input file")
    enc.add_argument("-p", "--password", required=True, help="Password for encryption")
    enc.add_argument("-o", "--output", required=True, help="Output encrypted file path")

    # Decrypt
    dec = subparsers.add_parser("decrypt")
    dec.add_argument("file", help="Path to encrypted file")
    dec.add_argument("-p", "--password", required=True, help="Password for decryption")
    dec.add_argument("-o", "--output", required=True, help="Output decrypted file path")

    # Integrity Check
    integ = subparsers.add_parser("integrity-check")
    integ.add_argument("file", help="Path to file to check or store")
    integ.add_argument("--store", action="store_true", help="Store hash instead of checking")

    # Demo
    subparsers.add_parser("demo")

    args = parser.parse_args()

    if args.command == "encrypt":
        encrypt_file(args.file, args.password, args.output)
    elif args.command == "decrypt":
        decrypt_file(args.file, args.password, args.output)
    elif args.command == "integrity-check":
        if args.store:
            hash_val = compute_hash(args.file)
            store_hash(args.file, hash_val)
            print(f"✅ Stored hash for {args.file}")
        else:
            check_integrity(args.file)
    elif args.command == "demo":
        demo()
    else:
        parser.print_help()

if __name__ == "__main__":
    main()
```