Lab 2 – AES Encryption with Buzzer

Learning Outcomes:

By the end of this lab, you should be able to:

1. Understand how AES (Advanced Encryption Standard) encrypts data in fixed-size 16-byte blocks.
2. Implement and use AES in **Electronic Code Book (ECB)** and **Cipher Block Chaining (CBC)** modes.
3. Compare how ECB leaks patterns while CBC provides stronger confidentiality.
4. Practice using Python with AES libraries and GPIO buzzer for physical feedback.
5. Recognize the importance of secure modes and initialization vectors (IVs) in real cryptographic systems.

Introduction to ECB and CBC

AES is a **block cipher**: it encrypts data in fixed-size 16-byte chunks.

Electronic Code Book (ECB):

- Encrypts each block independently.
- Identical plaintext blocks → identical ciphertext blocks.
- Easy to implement but leaks patterns (insecure for real data).

Cipher Block Chaining (CBC):

- Uses an **Initialization Vector (IV)** for the first block.
- Each plaintext block is XORed with the previous ciphertext block before encryption.
- Even repeated plaintext looks different in ciphertext.
- Stronger confidentiality, but requires secure IV handling and must not reuse IVs.

Lab Setup

Hardware

1. Raspberry Pi 5
2. Connect the Buzzer to the Raspberry Pi's GPIO pin
3. Don't forget to **ground** the Buzzer

Software

1. Python 3 (preinstalled on RPi)
2. PyCryptodome library for AES:

Copy & Paste the following into the Terminal for your Raspberry Pi one by one:

```
sudo apt-get update
sudo apt-get install -y python3 python3-pip
pip3 install pycryptodome pillow gpiozero
```

Starter Code Setup

You are provided with three Python files in your Lab 2 folder:

- aes.py – contains AES implementation in ECB and CBC modes.
- main.py – driver program that lets you encrypt/decrypt files and visualize images.
- buzzer.py – controls the buzzer on the Raspberry Pi.

Before starting the exercises, you must review and **complete small TODOs in the starter code (aes.py & buzzer.py)**.

DO NOT CHANGE ANYTHING IN MAIN.PY

Part 1 – Buzzer Feedback (10 points)

- The provided buzzer.py makes the Raspberry Pi beep on success or error.
- Open buzzer.py and complete fail and success functions
  - When encryption/decryption completes successfully → **3 short beeps**.
  - On failure (wrong key, padding error) → **1 long beep**.
- Run python3 -c "from buzzer import success,fail; success(); fail()" to test.

Part 2 – File Encryption with AES in ECB and CBC Modes (25 points)

In this part you will encrypt and decrypt a text file using AES in two different modes: **Electronic Code Book (ECB)** and **Cipher Block Chaining (CBC)**. You will compare their behavior, especially ciphertext lengths and security properties.

Step 1 - Complete the TO-DOs in aes.py

Open aes.py and complete the TODOs: (1) padding, (2) ECB encrypt & decrypt, (3) CBC encrypt & decrypt.

Step 2 – Prepare the test file

We will use a small plaintext file (test.txt) that is already provided in your lab folder.
The file should contain a short string, for example: HELLOECE371WELCOMETOLAB2

Step 3 – Encrypt the file with ECB

- Check the size of the file before and after encryption/decryption
- Run the following command:
  ```
  python3 main.py encrypt_ecb test.txt
  ```
- You will be prompted for a **16-character key**.
- After entering the key, the file test.txt will be overwritten with the ciphertext.
- Check the new file size:
  ```
  wc -c < test.txt
  ```

Step 4 – Decrypt the file with ECB

- Now restore the original plaintext:
  ```
  python3 main.py decrypt_ecb test.txt
  ```
- Reopen the file to confirm that the original string has returned

Step 5 – Encrypt the file with CBC

- Check the size of the file before and after encryption/decryption
- Run the following command:
  ```
  python3 main.py encrypt_cbc test.txt
  ```
- Enter the same 16-character key when prompted.
- Check the new file size:
  ```
  wc -c < test.txt
  ```

Step 6 – Decrypt the file with CBC

- Restore the plaintext again:
  ```
  python3 main.py decrypt_cbc test.txt
  ```
- Reopen the file to confirm that the original string has returned

Part 3 – Visualizing ECB vs CBC (25 points)

1. Place the given penguin.jpg in your lab folder.

Run visualization:

```
python3 main.py visualize penguin.jpg
```

2. Enter a 16-character key.
3. Two files will be created:
   - ecb.png
   - Cbc.png

Answer Questions(10 points):

1. Attach screenshots of the new files with labels (eg ecb.png and cbc.png):

2. Why is the CBC ciphertext longer than the ECB ciphertext?
3. Why does CBC require an Initialization Vector (IV) while ECB does not?
4. Which image still resembles the penguin? Which looks like noise? Why is ECB insecure for images?
5. Why is it dangerous to reuse an IV in CBC mode?
6. If an attacker flips bits in the ciphertext of CBC, what happens to the decrypted plaintext?

Deliverables

1. aes.py (with your completed TODOs)
2. buzzer.py (with your completed TODOs)
3. Screenshots of ECB and CBC outputs (ecb.png, cbc.png)
4. Written answers to all questions (Parts 2–4)
5. **Live Demo (30 Points)**