AES (Advanced Encryption Standard) Implementation Report

Suhas J 24MSRDF036

1. Introduction

AES (Advanced Encryption Standard) is a symmetric block cipher widely used for securing digital data. It encrypts data in 128-bit blocks using keys of 128, 192, or 256 bits. AES is fast, secure, and resistant to known cryptographic attacks.

Applications of AES:

- **Wi-Fi security (WPA2/WPA3)**
- **SSL/TLS for secure web communication**
- **Disk encryption tools (BitLocker, FileVault)**
- **Messaging apps (WhatsApp, Signal)**

2. AES Principles

- **Symmetric Cipher: Same key is used for encryption and decryption.**
- **Block Cipher: Operates on fixed-size blocks (128 bits).**
- **Rounds: AES-128 → 10 rounds, AES-192 → 12 rounds, AES-256 → 14 rounds.**

Feature	Caesar/Vigenère	AES
Key size	Small	128/192/256
Security	Weak	Strong
Operations	Shift/substitute	SubBytes, ShiftRows, MixColumns, AddRoundKey
Vulnerability	Frequency analysis	Practically infeasible attacks

3. Python Implementation

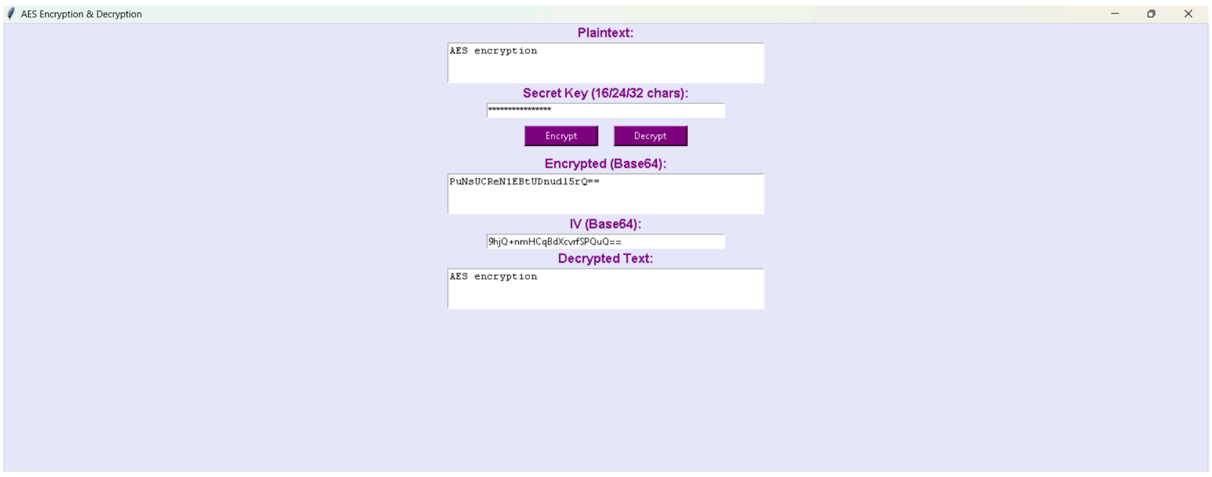
Library Installation:

```
pip install pycryptodome
```

```
import tkinter as tk
from tkinter import messagebox
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes


def encrypt_message():
    plaintext = entry_plaintext.get("1.0", tk.END).strip()
    key_input = entry_key.get().strip()

    if len(key_input) not in [16, 24, 32]:
        messagebox.showerror("Error", "Key must be 16, 24, or 32 
characters long")
        return

    key = key_input.encode()
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(plaintext.encode(), AES.block_size))

    ciphertext_b64 = b64encode(ct_bytes).decode()
    iv_b64 = b64encode(cipher.iv).decode()

    entry_cipher.delete("1.0", tk.END)
    entry_cipher.insert(tk.END, ciphertext_b64)

    entry_iv.delete(0, tk.END)
    entry_iv.insert(tk.END, iv_b64)


def decrypt_message():
    ciphertext_b64 = entry_cipher.get("1.0", tk.END).strip()
    iv_b64 = entry_iv.get().strip()
    key_input = entry_key.get().strip()
    if not ciphertext_b64 or not iv_b64:
        messagebox.showerror("Error", "Ciphertext and IV are required
for decryption")
        return

    if len(key_input) not in [16, 24, 32]:
        messagebox.showerror("Error", "Key must be 16, 24, or 32
characters long")
        return

    try:
        key = key_input.encode()
        ct = b64decode(ciphertext_b64)
        iv = b64decode(iv_b64)

        cipher = AES.new(key, AES.MODE_CBC, iv=iv)
        pt = unpad(cipher.decrypt(ct), AES.block_size)

        entry_decrypted.delete("1.0", tk.END)
        entry_decrypted.insert(tk.END, pt.decode())
    except Exception as e:
        messagebox.showerror("Error", f"Decryption failed: {e}")


# ---- GUI ----
root = tk.Tk()
root.title("AES Encryption & Decryption")
root.configure(bg="#E6E6FA")  # light lavender background

label_plaintext = tk.Label(root, text="Plaintext:", bg="#E6E6FA",
fg="purple", font=("Arial", 12, "bold"))
label_plaintext.pack()
entry_plaintext = tk.Text(root, height=3, width=50)
entry_plaintext.pack()

label_key = tk.Label(root, text="Secret Key (16/24/32 chars):",
bg="#E6E6FA", fg="purple", font=("Arial", 12, "bold"))
label_key.pack()
entry_key = tk.Entry(root, show="*", width=50)
entry_key.pack()
```

```
frame_buttons = tk.Frame(root, bg="#E6E6FA")
frame_buttons.pack(pady=10)

btn_encrypt = tk.Button(frame_buttons, text="Encrypt",
command=encrypt_message, bg="purple", fg="white", width=12)
btn_encrypt.grid(row=0, column=0, padx=10)

btn_decrypt = tk.Button(frame_buttons, text="Decrypt",
command=decrypt_message, bg="purple", fg="white", width=12)
btn_decrypt.grid(row=0, column=1, padx=10)

label_cipher = tk.Label(root, text="Encrypted (Base64):", bg="#E6E6FA",
fg="purple", font=("Arial", 12, "bold"))
label_cipher.pack()
entry_cipher = tk.Text(root, height=3, width=50)
entry_cipher.pack()

label_iv = tk.Label(root, text="IV (Base64):", bg="#E6E6FA",
fg="purple", font=("Arial", 12, "bold"))
label_iv.pack()
entry_iv = tk.Entry(root, width=50)
entry_iv.pack()

label_decrypted = tk.Label(root, text="Decrypted Text:", bg="#E6E6FA",
fg="purple", font=("Arial", 12, "bold"))
label_decrypted.pack()
entry_decrypted = tk.Text(root, height=3, width=50)
entry_decrypted.pack()

root.mainloop()
```

Code for the programme



Plaintext:

```
AES encryption
```

Secret Key (16/24/32 chars):

```
****************
```

Encrypted (Base64):

```
PuNsUCReN1EBtUDnudl5rQ==
```

IV (Base64):

```
9hjQ+nmHCqBdXcvrfSPQuQ==
```

Decrypted Text:

```
AES encryption
```

OUTPUT

Explanation of Input and Output

- **Input:**
  - **Plaintext:** Message to be encrypted (e.g., `"AES encryption"`).
  - **Secret Key:** Must be 16, 24, or 32 characters long.
  - **IV (Initialization Vector):** Randomly generated during encryption.
- **Output:**
  - **Encrypted Text (Base64):** Ciphertext shown in Base64 for readability.
  - **IV (Base64):** Shown so that decryption can use the same IV.
  - **Decrypted Text:** Original plaintext after decryption.

Imported Modules:

- `Crypto.Cipher.AES` → AES encryption/decryption
- `Crypto.Util.Padding` → Adds/removes padding
- `Crypto.Random` → Generates random IV
- `base64` → Converts ciphertext/IV into readable format
- `tkinter` → GUI interface

4. Summary

This project demonstrates **AES encryption & decryption** using a Tkinter-based GUI.

- It accepts a plaintext message and a secret key (16/24/32 chars).
- Encrypts text using **AES in CBC mode**, generating a random IV.
- Outputs ciphertext and IV in Base64 format.
- Supports decryption back to original plaintext.
- Uses **PyCryptodome** library for cryptographic operations.

5. References

1. PyCryptodome Documentation: https://www.pycryptodome.org
2. NIST AES Specification: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
3. Practical Cryptography Guide: https://cryptography.io