# AES Encryption and Decryption Implementation in Python

## 1. Introduction

The Advanced Encryption Standard (AES) is a widely used symmetric block cipher algorithm standardized by NIST in 2001. AES replaced the older Data Encryption Standard (DES) due to DES's vulnerability to brute-force attacks. AES is considered highly secure and is used in modern communication systems, banking, government, and everyday applications like Wi-Fi and HTTPS.AES operates on fixed 128-bit (16-byte) blocks of plaintext and supports three key sizes: AES-128 (16 bytes), AES-192 (24 bytes), AES-256 (32 bytes).

## 2. AES Block Cipher Logic

AES is a symmetric block cipher, meaning the same key is used for encryption and decryption. It works on 128-bit blocks and involves multiple rounds of substitution, permutation, and mixing operations.

The number of rounds depends on the key size:

- AES-128 → 10 rounds

- AES-192 → 12 rounds

- AES-256 → 14 rounds

Modes of operation (like ECB, CBC, CFB, OFB, and GCM) are used to encrypt larger texts securely. CBC(cipher block chaining) is the most common in SSL/TLS.

## 3. Implementation in Python

We used the PyCryptodome library for AES in CBC mode with PKCS7 padding. Tk library for GUI.The secret key must be 16/24/32 bytes. An Initialization Vector (IV) ensures unique ciphertext for the same plaintext.A Tkinter-based GUI in lavender theme is provided for user-friendly encryption and decryption.

## 4. Input/Output Explanation

Example:

Input Plaintext: "hello,i am monisha"

Secret Key: "mysecretpassword" (16 bytes → AES-128)

Output Ciphertext: CIJRy... (Base64)

Decrypted Plaintext: "hello,i am monisha"

## 5. Comparison with Classical Ciphers

### AES vs Classical Ciphers (Caesar, Vigenère)

1. **Security Level**

- **AES**: Strong, modern, resistant to brute-force attacks.
- **Caesar/Vigenère**: Weak, easily breakable with frequency analysis or brute force.

2. **Key Size**

- **AES**: 128, 192, or 256 bits (huge key space).
- **Caesar**: Only 25 possible keys.**Vigenère**: Slightly better than Caesar, but still weak for short keys.

3. **Block Cipher vs Substitution**

- **AES**: Block cipher (operates on 128-bit blocks with substitution–permutation network).
- **Caesar/Vigenère**: Simple substitution ciphers (replace letters with shifted/mapped ones).

4. **Real-World Usage**

- **AES**: Used in Wi-Fi security (WPA2), SSL/TLS (HTTPS), VPNs, disk encryption.
- **Caesar/Vigenère**: No real-world security usage today, only educational or puzzle contexts.

5. **Attack Resistance**

- **AES**: Designed to resist cryptanalysis and brute-force; practically unbreakable with current computers.
- **Caesar/Vigenère**: Broken within seconds using basic cryptanalysis techniques.

## 6. Real-world Applications of AES

- Wi-Fi Security (WPA2/WPA3)

- Banking Transactions

- TLS/SSL (HTTPS)

- File Encryption (7-Zip, VeraCrypt, BitLocker)

- Cloud Storage Security (Google Drive, Dropbox)

- Messaging Apps (WhatsApp, Signal, Telegram)

## 7. Installation Guide

1. Install Python (>=3.8)

2. Install PyCryptodome:

```
pip install pycryptodome
```

3. For GUI (Tkinter):

```
pip install tk
```

## 8. Security Analysis

- AES is considered secure against all known practical attacks.
- Brute-forcing AES-128 would take billions of years with current technology.
- AES-256 remains secure even against potential quantum computing threats (Grover's algorithm).

## 9. Conclusion

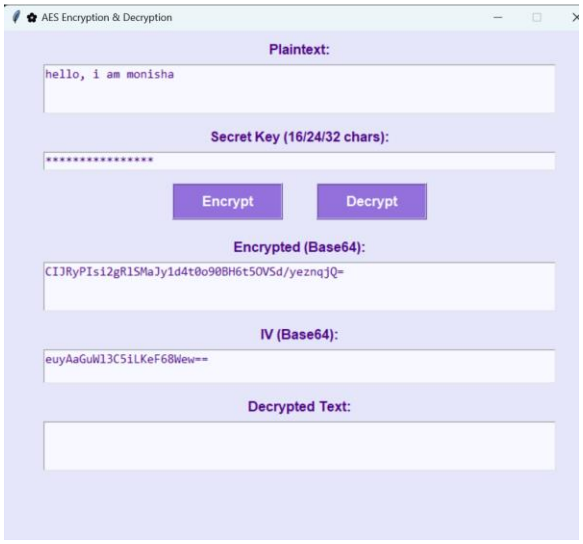
AES is a secure, efficient, and reliable encryption algorithm. The Python implementation with GUI demonstrates encryption and decryption with proper key size, padding, and block mode handling. Compared to classical ciphers like Caesar or Vigenère, AES is vastly superior in security and practical applications.

## 10. Code

```
import tkinter as tk

from tkinter import messagebox

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

import base64


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))


    iv = base64.b64encode(cipher.iv).decode()

    ciphertext = base64.b64encode(ct_bytes).decode()
    entry_encrypted.delete("1.0", tk.END)
    entry_encrypted.insert(tk.END, ciphertext)

    entry_iv.delete("1.0", tk.END)
    entry_iv.insert(tk.END, iv)

def decrypt_message():
    ciphertext = entry_encrypted.get("1.0", tk.END).strip()
    key_input = entry_key.get().strip()
    iv = entry_iv.get("1.0", tk.END).strip()

    try:
        key = key_input.encode()
        cipher = AES.new(key, AES.MODE_CBC, base64.b64decode(iv))
        pt = unpad(cipher.decrypt(base64.b64decode(ciphertext)), AES.block_size)
        decrypted = pt.decode()

        entry_decrypted.delete("1.0", tk.END)
        entry_decrypted.insert(tk.END, decrypted)

    except Exception as e:
        messagebox.showerror("Error", f"Decryption failed: {str(e)}")


# --- GUI Setup ---

root = tk.Tk()

root.title("🌸 AES Encryption & Decryption")

root.geometry("700x650")

root.configure(bg="#E6E6FA")  # Lavender background

root.resizable(False, False)


# --- Styles ---

label_style = {"bg": "#E6E6FA", "fg": "#4B0082", "font": ("Arial", 12, "bold")}

text_style = {"height": 3, "width": 75, "bg": "#F8F8FF", "fg": "#4B0082", "font": ("Consolas", 11)}

entry_style = {"width": 75, "bg": "#F8F8FF", "fg": "#4B0082", "font": ("Consolas", 11)}


button_style = {

    "bg": "#9370DB",   # Purple button

    "fg": "white",

    "font": ("Arial", 12, "bold"),

    "activebackground": "#BA55D3",

    "activeforeground": "white",

    "relief": "ridge",
    "width": 12,
    "pady": 6
}

# --- Widgets ---
tk.Label(root, text="Plaintext:", **label_style).pack(pady=(10, 0))
entry_plaintext = tk.Text(root, **text_style)
entry_plaintext.pack(pady=5)

tk.Label(root, text="Secret Key (16/24/32 chars):", **label_style).pack(pady=(10, 0))
entry_key = tk.Entry(root, **entry_style, show="*")
entry_key.pack(pady=5)

frame_btn = tk.Frame(root, bg="#E6E6FA")
frame_btn.pack(pady=10)
tk.Button(frame_btn, text="Encrypt", command=encrypt_message, **button_style).grid(row=0,
column=0, padx=20)
tk.Button(frame_btn, text="Decrypt", command=decrypt_message, **button_style).grid(row=0,
column=1, padx=20)

tk.Label(root, text="Encrypted (Base64):", **label_style).pack(pady=(10, 0))
entry_encrypted = tk.Text(root, **text_style)
```

```
entry_encrypted.pack(pady=5)


tk.Label(root, text="IV (Base64):", **label_style).pack(pady=(10, 0))

entry_iv = tk.Text(root, height=2, width=75, bg="#F8F8FF", fg="#4B0082", font=("Consolas", 11))

entry_iv.pack(pady=5)


tk.Label(root, text="Decrypted Text:", **label_style).pack(pady=(10, 0))

entry_decrypted = tk.Text(root, **text_style)

entry_decrypted.pack(pady=5)


root.mainloop()
```

## 11. Screenshots


AES Encryption & Decryption

Plaintext:

```
hello, i am monisha
```

Secret Key (16/24/32 chars):

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

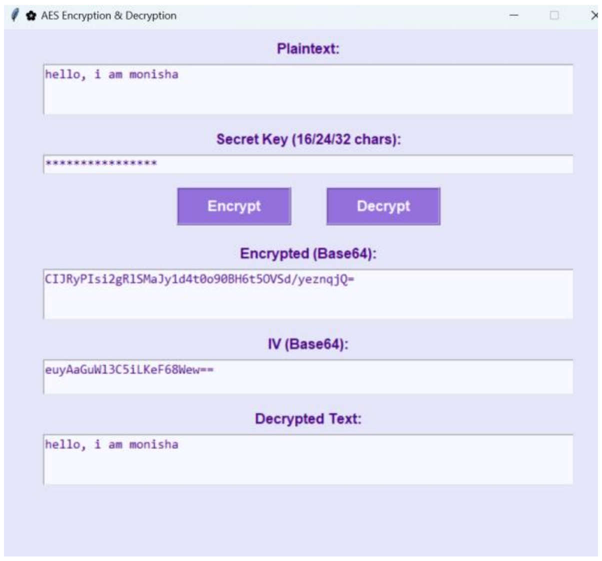
Encrypted (Base64):

```
CIJRyPIsi2gRlSMaJy1d4t0o90BH6t5OVSd/yeznqjQ=
```

IV (Base64):

```
euyAaGuWl3C5iLKeF68Wew==
```

Decrypted Text:


AES Encryption & Decryption

Plaintext:

```
hello, i am monisha
```

Secret Key (16/24/32 chars):

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

Encrypted (Base64):

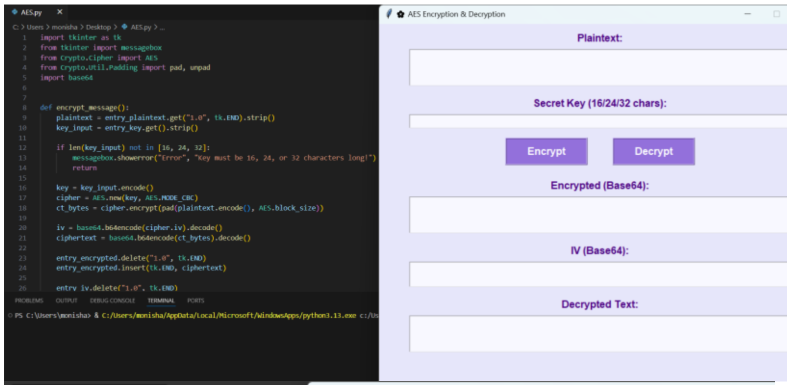
```
CIJRyPIsi2gRlSMaJy1d4t0o90BH6t5OVSd/yeznqjQ=
```

IV (Base64):

```
euyAaGuWl3C5iLKeF68Wew==
```

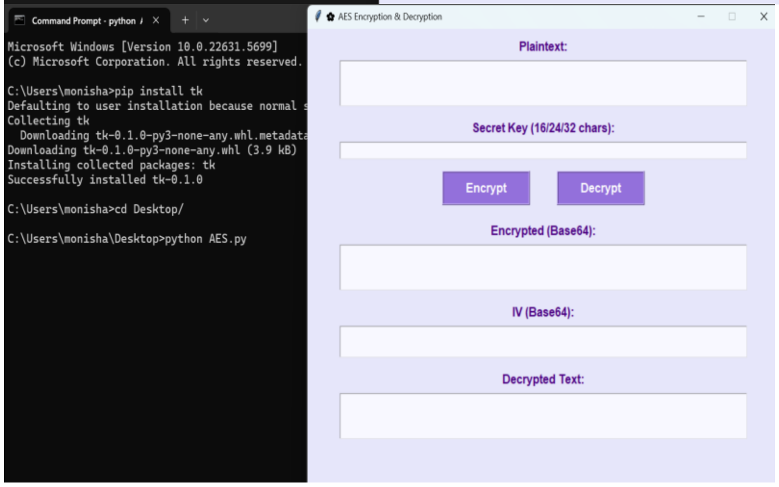
Decrypted Text:

```
hello, i am monisha
```



```python
import tkinter as tk
from tkinter import messagebox
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64


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))

    iv = base64.b64encode(cipher.iv).decode()
    ciphertext = base64.b64encode(ct_bytes).decode()

    entry_encrypted.delete("1.0", tk.END)
    entry_encrypted.insert(tk.END, ciphertext)

    entry_iv.delete("1.0", tk.END)
```

```
PS C:\Users\monisha> & C:/Users/monisha/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Us
```



```
Microsoft Windows [Version 10.0.22631.5699]
(c) Microsoft Corporation. All rights reserved.

C:\Users\monisha>pip install tk
Defaulting to user installation because normal s
Collecting tk
  Downloading tk-0.1.0-py3-none-any.whl.metadata
Downloading tk-0.1.0-py3-none-any.whl (3.9 kB)
Installing collected packages: tk
Successfully installed tk-0.1.0

C:\Users\monisha>cd Desktop/

C:\Users\monisha\Desktop>python AES.py
```

## 12. Reference

- PyCryptodome Documentation →https://pycryptodome.readthedocs.io
- NIST AES Standard (FIPS 197)