```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk

# Load CIFAR-10 dataset
(x_train, _), (x_test, _) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize images

# Encryption Key (Used for XOR transformation)
encryption_key = np.random.rand(128)  # Fixed-length key

# Define Encoder Model
def create_encoder():
    encoder = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2), padding='same'),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), padding='same'),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), padding='same'),
        layers.Flatten(),
        layers.Dense(128, activation='relu')  # Bottleneck (compressed representation)
    ])
    return encoder

# Define Decoder Model
def create_decoder():
    decoder = models.Sequential([
        layers.Dense(4 * 4 * 128, activation='relu'),
        layers.Reshape((4, 4, 128)),
        layers.Conv2DTranspose(128, (3, 3), activation='relu', padding='same'),
        layers.UpSampling2D((2, 2)),
        layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same'),
        layers.UpSampling2D((2, 2)),
        layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same'),
        layers.UpSampling2D((2, 2)),
        layers.Conv2DTranspose(3, (3, 3), activation='sigmoid', padding='same')
    ])
    return decoder

# Create models
encoder = create_encoder()
decoder = create_decoder()
```

```
# Full Model: Autoencoder
input_img = tf.keras.Input(shape=(32, 32, 3))
encoded = encoder(input_img)
decoded = decoder(encoded)
autoencoder = tf.keras.Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(x_train, x_train, epochs=10, batch_size=64, validation_data=(x_test, x_test))

# Encryption Function (Applying XOR with key)
def encrypt_image(idx):
    global encrypted_img
    img = x_test[idx]
    encoded_img = encoder.predict(np.expand_dims(img, axis=0))[0]

    # Apply XOR-like transformation with key
    encrypted_img = encoded_img * encryption_key
    display_images(img, encrypted_img.reshape((8, 16)), "Encrypted (XOR Applied)")

# Decryption Function
def decrypt_image():
    global encrypted_img
    decrypted_encoded = encrypted_img / encryption_key  # Reverse XOR operation
    decrypted_img = decoder.predict(np.expand_dims(decrypted_encoded, axis=0))[0]
    display_images(encrypted_img.reshape((8, 16)), decrypted_img, "Decrypted")

# Display Original and Processed Images
def display_images(original, processed, title):
    fig, axes = plt.subplots(1, 2, figsize=(6, 3))
    axes[0].imshow(original)
    axes[0].set_title("Original")
    axes[0].axis('off')
    axes[1].imshow(processed)
    axes[1].set_title(title)
    axes[1].axis('off')
    plt.show()

# GUI for Image Selection
def show_sample_images():
    def on_select(event):
        idx = int(event.widget.get(event.widget.curselection()))
        encrypt_image(idx)

    root = tk.Tk()
    root.title("CIFAR-10 Image Encryption")
    frame = ttk.Frame(root, padding=10)
    frame.grid(row=0, column=0)
    label = ttk.Label(frame, text="Select an image index to encrypt:")
```

```
    label.grid(row=0, column=0)
    listbox = tk.Listbox(frame, height=10)
    listbox.grid(row=1, column=0)
    for i in range(10):
        listbox.insert(tk.END, str(i))
    listbox.bind('<<ListboxSelect>>', on_select)

    decrypt_button = ttk.Button(frame, text="Decrypt Image", command=decrypt_image)
    decrypt_button.grid(row=2, column=0)
    root.mainloop()

# Run GUI
show_sample_images()
```