# PROJECT - REPORT

## Image Encryption Using Triple-DES Algorithm

**NAME- SOURABH KUMAR**

**INTERNSHIP ROLE - CYBERSECURITY DEVELOPER**

**ORGANISATION- EXPOSYS DATA LAB**

**COLLEDGE-THE NATIONAL INSTITUTE OF ENGINEERING MYSORE**

# Abstract

Nowadays the use of devices such as computers, mobile and many more other devices for communication as well as for data storage and transmission has increased. As a result, there is an increase in the number of users. Along with these users, there is also an increase in the number of unauthorized users who are trying to access data by unfair means. This arises the problem of data security. Images are sent over an insecure transmission channel from different sources, some image data contains secret data, some images themselves are highly confidential hence, securing them from any attack is essentially required. To solve this problem, we are using the Triple-DES algorithm for encrypting and decrypting images. This encrypted data is unreadable to unauthorized users. This encrypted data can be sent over the network and can be decrypted using Triple-DES at the receiving end. Hence it ensures secure transmission of the image.

This is achieved using the ImageEncrypto app which working on Triple-DES Algorithm to encrypt an image with a secure key provided by the user and when the receiver gets access to the encrypted image they can Decrypt them with the help of a valid key known to the user and the receiver only.

**TABLE OF CONTENTS**

# INTRODUCTION

In today's world almost all digital services like internet communication, medical military imaging systems, the multimedia system needs high-level security. There is a need for a security level in order to safely store and transmit digital images containing critical information. This is because of the faster growth in multimedia technology, the internet, and cell phones.
Therefore there is a need for image encryption techniques in order to hide images from such attacks.
In this system, we use Triple DES (Data Encryption Standard) in order to hide images from intruders. Such an Encryption technique helps to avoid intrusion attacks. So, Image Encrypto is an application that allows the users to encrypt images using the triple-DES algorithm to convert the byte form of the image to encrypted cipher for which can then be stored in a different file format and then be sent around the globe to the receiver where only he can decrypt it with the key.

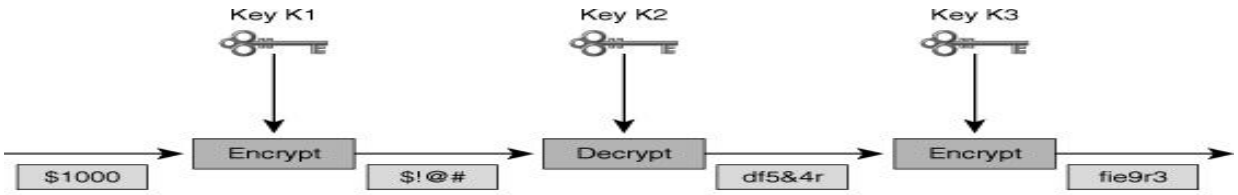
# FEATURES

- Encryption of an Image to unreadable format.
- The decryption of encrypted image to the original image.
- Secure transfer of an image over the network such as the internet
- Ensure no modifications are made while transferring over the network.

## Scope of the project

### Product scope

The project works by encrypting the given image using the Triple-DES algorithm so that this image can be sent securely over the network. On the receiver side, the receiver has code for decrypting the image so that he can get the original image. This helps in sending confidential and sensitive information securely over the internet. The main application of this can be very helpful in medical and military fields.




- EDE (Encrypt-Decrypt-Encrypt) Method – 3DES-EDE Method:
  - Data is encrypted using K1.
  - Data is decrypted using K2.
  - Data is encrypted using K3.
- If K1 = K3, Key Yields 112-Bit Key Length
- If K1 ≠ K3, Key Yields 168-Bit Key Length

### Design and Implementation constraints

- Python must be used for front end
- Encryption and Decryption should be done using the Triple-DES algorithm
- Original Image must be in .jpeg/.png format.

## METHODOLOGY

The architecture of the application consists of the back end and the front end and Firebase real-time database, both of them having their own set dependencies (libraries and frameworks). The front end is the presentation layer that the end-user sees when they enter the application. The back end provides all the data and part of the logic and it is running behind the scenes.

**BACK END:**

"Back end" refers to the logic and data layers running on the application. In our case, the back end makes sure that the data introduced through the client application (the front end), is valid. Since the front end can be avoided or easily manipulated (the source code is available to the end-user) we have to make sure that all the requests we receive are first verified by the back end, the user has the appropriate permissions, the parameters are valid, etc. The Backend language used here is PYTHON.

**MODELS USED IN THE APPLICATION:**

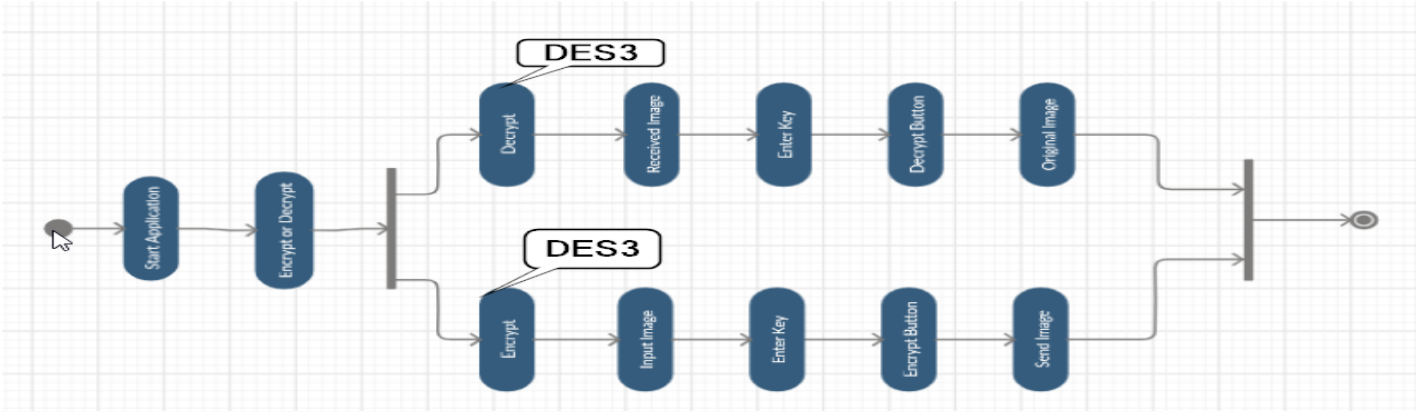
Triple DES MODEL: (or TDES or TDEA or 3DES) is a symmetric block cipher standardized by NIST in SP 800-67 Rev1, though they will deprecate it soon.

TDES has a fixed data block size of 8 bytes. It consists of the cascade of 3 Single DES ciphers (EDE: Encryption - Decryption - Encryption), where each stage uses an independent DES sub-key. The standard defines 3 *Keying Options*:

- *Option 1*: all sub-keys take different values (parity bits ignored). The TDES key is therefore 24 bytes long (concatenation of *K1*, *K2*, and *K3*) , to achieve 112 bits of effective security.

- *Option 2*: *K1* matches *K3* but *K2* is different (parity bits ignored). The TDES key is 16 bytes long (concatenation of *K1* and *K2*), to achieve 90 bits of effective security. In this mode, the cipher is also termed 2TDES.
- *Option 3*: *K1 K2*, and *K3* all match (parity bits ignored). As result, Triple DES degrades to Single DES.

**Activity Diagram:**




**DEMO CODE OF DES3 USING THE CRYPTO CIPHER MODULE:**

```
>>> from Crypto.Cipher import DES3

>>> from Crypto.Random import get_random_bytes

>>> while True:

>>>     try:

>>>         key = DES3.adjust_key_parity(get_random_bytes(24))
```

```
>>>         break

>>>     except ValueError:

>>>         pass

>>> cipher = DES3.new(key, DES3.MODE_CFB)

>>> plaintext = b'We are no longer the knights who say ni!'

>>> msg = cipher.iv + cipher.encrypt(plaintext)
```

This module is used to encrypt the image with Key1 and then the ciphertext is decrypted with key2 and the decrypted text is then encrypted with the key3, finally forming the encrypted ciphertext.

## FRONT END:

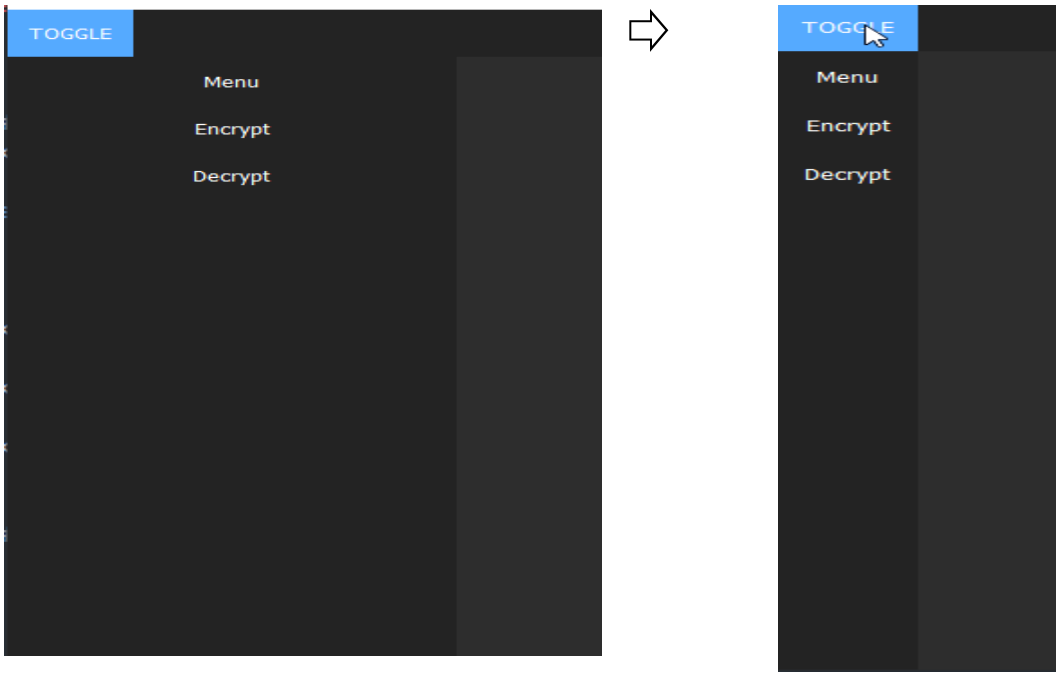
When it comes to the front end the user interface of any application is really very important in terms of how user-friendly it is an eye it is. If the UI of an app is not appealing to users then however the back end and logic must be really well executed the app will not be used by any user.

So, keeping this in mind I have built the UI with XML markup Language supported in Qt Designer and some really eye-catching and meaningful Animations. The UI is made so that the user can easily navigate the app without consulting the developers or anyone else. Vector Assets have been used with the proper color combinations. There has been using of certain modules also like the following-:

- ## TOGGLE/BURGUER MENU

1. self.ui.Btn_Toggle.clicked.connect(lambda: UIFunctions.toggleMenu(self, 250, True))

   This is used to create a custom toggle menu for the user.




- ## MAIN MENU

2. self.label_1.setText(QCoreApplication.translate("MainWindow", u"Image Encryptor", None))

This is the main menu for the user.

- ## ENCRYPT MENU

3. self.label_2.setText(QCoreApplication.translate("MainWindow", u"Encrypt Image", None))

The Encryption page for the app.


- ## DECRYPT MENU

4. self.label_4.setText(QCoreApplication.translate("MainWindow", u"Decrypt Image", None))

The Decryption page for the app.

# IMPLEMENTATION

The whole project is done in Qt Designer version 5.14.1 and sublime text Build 4107.

IMPORTANT ACTIVITIES:

1. Main window

The main screen id created using the pyside2 module of python and this is the code snippet.

```
import sys
import os
import platform
from PySide2 import QtCore, QtGui, QtWidgets
from PySide2.QtCore import (QCoreApplication, QPropertyAnimation, QDate, QDateTime, QMetaObject, QObject, QPoint, QRect, QSize, QT
from PySide2.QtGui import (QBrush, QColor, QConicalGradient, QCursor, QFont, QFontDatabase, QIcon, QKeySequence, QLinearGradient,
from PySide2.QtWidgets import *
from Encrypt_Decrypt import Encrypter, Decrypter
import base64

# GUI FILE
from ui_main import Ui_MainWindow
# IMPORT FUNCTIONS FOR TOGGLE
from ui_functions import *
```

These are the modules and submodules used to construct the main page.

The main page code snippet:

```
class MainWindow(QMainWindow, encrypt_page,decrypt_page):
    def __init__(self):
        QMainWindow.__init__(self)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        encrypt_page.__init__(self)
        decrypt_page.__init__(self)


        ## TOGGLE/BURGUER MENU
        ########################################################################
        self.ui.Btn_Toggle.clicked.connect(lambda: UIFunctions.toggleMenu(self, 250, True))
        self.Handel_Buttons()
        self.ui.stackedWidget.setCurrentWidget(self.ui.page_1)
    def Handel_Buttons(self):

        # PAGE 1
        self.ui.btn_page_1.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.page_1))
        # PAGE 2
        self.ui.btn_page_2.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.page_2))
        # PAGE 3
        self.ui.btn_page_3.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.page_3))

        ## SHOW ==> MAIN WINDOW
        ########################################################################
        self.show()
        ## ==> END ##

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    sys.exit(app.exec_())
```

The Encryption Window is brought up using the btn_page_2 key, which transmits the command to the Encrypt_page function where the Encryption process begins.

2. The encryption_page function

The code snippet:

```
# ENcrypting the Image
class encrypt_page():
    def __init__(self):
        self.file={}
        self.string=""
        self.Handel_Buttons()
        # selecting the image
        self.ui.btn_file_choose.clicked.connect(self.chooseFile)
        self.ui.btn_encrypt.clicked.connect(self.onClickEncrypt)
    def Handel_Buttons(self):
        self.ui.btn_page_2.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.page_2))
    def chooseFile(self):
        self.file= QFileDialog.getOpenFileName(self, "Load Image")
        pixmap = QtGui.QPixmap(self.file[0])
        self.ui.img_lbl.setPixmap(pixmap.scaledToHeight(201))
        if self.file != None:
            ba = QtCore.QByteArray()
            buff = QtCore.QBuffer(ba)
            buff.open(QtCore.QIODevice.WriteOnly)
            try:
                ok = pixmap.save(buff, "PNG")
                assert ok
            except AssertionError:
                self.message="No Image Selected"
                self.show_popup()
                return
            pixmap_bytes = ba.data()
            self.string = base64.b64encode(pixmap_bytes)
```

Moving from here the image bytes will be transferred to the on_clickedEncrypt function

```
def onClickEncrypt(self):
    mykey = self.ui.line_key.text()
    if not mykey:
        self.message="Enter a key"
        self.show_popup()
        return
    if not self.string:
        self.message="Select an Image"
        self.show_popup()
        return
    x = Encrypter(self.string, mykey)
    cipher = x.encrypt_image()
    if not os.path.exists("EncryptedFiles"):
        os.makedirs("EncryptedFiles")
    os.chdir(os.getcwd() + r"\EncryptedFiles")
    fh = open("cipher.txt", "wb")
    fh.write(cipher)
    fh.close()
    self.message = "Encryption Successfull: file strored in \n"+ os.getcwd()
    self.show_popup()
```

In the onClickedEncrypt function, the user key is first checked and the image file data is checked, if failed to validate the above data the application will popup a necessary message to the user regarding the error caused.

The code snippet to the error message popup window:

```
def show_popup(self):
    font = QFont()
    font.setPointSize(10)
    msg = QMessageBox()
    if self.message == "Enter a key" or self.message == "Select an Image" or self.message=="No Image Selected":
        msg.setIcon(QMessageBox.Warning)
        msg.setWindowTitle("Warning")
    else:
        msg.setIcon(QMessageBox.Information)
        msg.setWindowTitle("Result")
    msg.setFont(font)
    msg.setText(self.message)
    msg.setStyleSheet(u"background-color: rgb(35, 35, 35) ; color : white;")
    msg.setStandardButtons(QMessageBox.Ok)
    x = msg.exec_()
```

If all correct the program will be transferred to the Encryp_Decrypt class where the data will be further processed and separated according to the request.

2.1Encrypt_Decrypt Class:

In Encrypter the user key from the user and the byte data of the image is collected and the data is passed to the DES3cipher class where the actual encryption of the image is done.

```
class Encrypter:
    def __init__(self, text,key):
        self.text = text
        self.key =  key
    def encrypt_image(self):
        des = DESCipher(self.key)
        cipher = des.encrypt(self.text)
        return cipher
```

Next, the user key is shifted the converted to a 32-byte string using the SHA256 Algorithm and then the key is trimmed to the first 16 bytes of the 32 bytes string

Using the list selection method.

```
def __init__(self, key):
    self.bs = DES3.block_size
    self.key = hashlib.sha256(key.encode()).digest()
    self.key = self.key[16:]
```

Next, the text is forwarded to the encrypt function in the DES3Cipher class where the byte The format of the image is ciphered to the Des3 by passing it first to the padding section where the byte is properly padded before the actual encryption.

```
def encrypt(self, raw):
    raw = self._pad(raw)
    iv = Random.new().read(DES3.block_size)
    cipher = DES3.new(self.key, DES3.MODE_CBC, iv)
    return base64.b64encode(iv + cipher.encrypt(raw))
```

⇩ the Text is padded and then passed to the function ⇧

```
def _pad(self, s):
    return s + (self.bs - len(s) % self.bs) * chr(self.bs - len(s) % self.bs).encode('utf-8')
```

Then the cipher image is then transmitted to the previous Encrypter class and then to the onClickEncrypt Function where the cipher data is stored in the Cipher.txt file.

3. Decryption_page function

_____In this function, the ciphertext is first examined, and the key is then passed for processing. The ciphertext is further checked for validity and then encrypted to “UTF-8” Format.

The code snippet:

```
# Decrypting the image
class decrypt_page():
    def __init__(self):
        self.cipher = {}
        self.Handel_Buttons()
        # selecting the cipher text
        self.ui.btn_file_choose_2.clicked.connect(self.chooseFile1)
        self.ui.btn_decrypt.clicked.connect(self.onClickDecrypt)
    # moving to the
    def Handel_Buttons(self):
        self.ui.btn_page_3.clicked.connect(
            lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.page_3))

    def chooseFile1(self):
        file = QFileDialog.getOpenFileName(self, 'Open File')
        try:
            text = open(file[0]).read()
        except FileNotFoundError:
            self.message="No file Selected"
            self.show_popup()
            return
        self.cipher = text.encode('utf-8')
```

Moving from here the image bytes will be transferred to the on_clickedDecrypt function

```
def onClickDecrypt(self):
    mykey = self.ui.line_key_2.text()
    if not mykey:
        self.message="Enter a key"
        self.show_popup()
        return
    if not self.cipher:
        self.message="Select an cipher.txt File"
        self.show_popup()
        return
    x = Decrypter(self.cipher)
    image = x.decrypt_image(mykey)

    ba = QtCore.QByteArray(image)
    pixmap = QtGui.QPixmap()
    ok = pixmap.loadFromData(ba, "PNG")
    assert ok
    self.ui.img_lbl_2.setPixmap(pixmap.scaledToHeight(201))
    self.message = "Decryption Successfull: file strored in \n"+ os.getcwd()
    self.show_popup()
```

In the onClickedDecrypt function, the user key is first checked and the image file data is checked, if failed to validate the above data the application will popup a necessary message to the user regarding the error caused.

The code snippet to the error message popup window:

```
def show_popup(self):
    font = QFont()
    font.setPointSize(10)
    msg = QMessageBox()
    if self.message == "Enter a key" or self.message == "Select an cipher.txt File" or self.message == "No file Selected":
        msg.setIcon(QMessageBox.Warning)
        msg.setWindowTitle("Warning")
    else:
        msg.setIcon(QMessageBox.Information)
        msg.setWindowTitle("Result")
    msg.setFont(font)
    msg.setText(self.message)
    msg.setStyleSheet(u"background-color: rgb(35, 35, 35) ; color : white;")
    msg.setStandardButtons(QMessageBox.Ok)
    x = msg.exec_()
```

If all correct the program will be transferred to the Encryp_Decrypt class where the data will be further processed and separated according to the request.

3.1Encrypt_Decrypt Class:

In Decrypter the user key from the user and the byte data of the image is collected and the data is passed to the DES3cipher class where the actual decryption of the ciphertext is done.

```
class Decrypter:
    def __init__(self, cipher):
        self.cipher = cipher
    def decrypt_image(self,k):
        key = k
        cipher = self.cipher
        des = DESCipher(key)
        base64_decoded = des.decrypt(cipher)
        fh = open("decryptedImage.png", "wb")
        fh.write(base64.b64decode(base64_decoded))
        fh.close()
        return (base64.b64decode(base64_decoded))
```

Next, the user key is shifted the converted to a 32-byte string using the SHA256 Algorithm and then the key is trimmed to the first 16 bytes of the 32 bytes string

Using the list selection method.

```
def __init__(self, key):
    self.bs = DES3.block_size
    self.key = hashlib.sha256(key.encode()).digest()
    self.key = self.key[16:]
```

Next, the text is forwarded to the decrypt function in the DES3Cipher class where the byte The format of the ciphertext is deciphered from the Des3 format to byte form by passing it first to the unpadding section where the byte is properly padded before the actual decryption.

```
def decrypt(self, enc):
    enc = base64.b64decode(enc)
    iv = enc[:DES3.block_size]
    cipher = DES3.new(self.key, DES3.MODE_CBC, iv)
    return self._unpad(cipher.decrypt(enc[DES3.block_size:])).decode('utf-8')
```

⇩ The Cipher is unpadded and then passed to the function ⇧

```
@staticmethod
def _unpad(s):
    return s[:-ord(s[len(s)-1:])]
```

Then the decipher image is then transmitted to the previous Decrypter class and then to the onClickDecrypt Function where the cipher data is stored in the decryptedImage.png.

# CONCLUSION

There is always someplace for enhancements in any software application, however good and efficient the application may be. Right now, we are dealing with only instant messaging between peers.

We have successfully developed a program that encrypts and decrypts the image images accurately. This will help in minimizing the problem of data theft and leaks of other sensitive information. The file that we obtained after encryption is very safe and no one can steal data from this file. So, this file can be sent on a network without worrying. Our developed solution is a small contribution that can be very helpful for military or medical fields in future times.