

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

Nếu chúng ta đã quen thuộc với các công nghệ tạo ứng dụng Desktop truyền thống như Winform, WPF (C#) hoặc Swing, JavaFx (Java), thì trong bài thực hành này, chúng ta sẽ khám phá cách sử dụng các công cụ hỗ trợ phát triển giao diện desktop cho ứng dụng bảo mật bằng Python, bao gồm PyQt5 và QtDesigner. Đồng thời, chúng ta cũng sẽ tìm hiểu về hệ mã hóa RSA, đường cong Elliptic và cách ứng dụng chúng trong việc phát triển các phần mềm bảo mật.

3.1 PYQT5

PyQt là một thư viện giúp chúng ta truy cập Qt GUI, một framework nổi tiếng được phát triển bằng C++. Trong số nhiều phiên bản của PyQt, phiên bản mới nhất và phổ biến nhất hiện nay là PyQt5. Thư viện này hỗ trợ người dùng xây dựng các ứng dụng desktop có giao diện đồ họa mạnh mẽ bằng Python, kết hợp khả năng lập trình linh hoạt của Python với các công cụ và tính năng phong phú từ Qt5. Các đặc điểm chính của PyQt5 bao gồm:

- Widget và Layout: PyQt5 cung cấp rất nhiều loại widget (đối tượng giao diện) để xây dựng các thành phần giao diện người dùng như nút, ô nhập liệu, menu,....
- Signals và Slots: PyQt5 sử dụng hệ thống "signals and slots" để xử lý sự kiện và tương tác giữa các thành phần.
- Tích hợp với Python: PyQt5 được viết bằng ngôn ngữ C++ và Python, nhưng API được thiết kế sao cho người dùng có thể dễ dàng sử dụng từ Python.
- Đa nền tảng: Ứng dụng được tạo bằng PyQt5 có thể chạy trên nhiều nền tảng khác nhau.

- Phong phú với Qt Designer: PyQt5 đi kèm với Qt Designer, một công cụ đồ họa cho phép thiết kế giao diện người dùng bằng cách kéo và thả các widget.

3.2 QTDESIGNER

Qt Designer là một công cụ được cung cấp bởi Qt Toolkit, được sử dụng để thiết kế giao diện người dùng đồ họa (GUI) cho ứng dụng sử dụng Qt. Một số điểm quan trọng về Qt Designer bao gồm:

- Giao diện đồ họa dễ sử dụng: Qt Designer cho phép người dùng thiết kế giao diện người dùng một cách trực quan thông qua giao diện đồ họa.
- Tích hợp tốt với PyQt và Qt: Qt Designer có thể được sử dụng để thiết kế giao diện cho các ứng dụng sử dụng PyQt framework hoặc bất kỳ ứng dụng Qt.
- Công cụ kéo và thả (Drag-and-Drop): Điểm mạnh của Qt Designer là khả năng sử dụng công cụ kéo và thả.
- Tạo mã nguồn tự động: Qt Designer có khả năng tạo mã nguồn tự động dựa trên các thiết kế giao diện người dùng.

3.3 HỆ MÃ HÓA RSA

3.3.1 Giới thiệu RSA

"RSA là một phương pháp mã hóa và giải mã dựa trên cặp khóa công khai (cryptography public-key), được sáng tạo bởi ba nhà khoa học Ronald Rivest, Adi Shamir và Leonard Adleman vào những năm 1970. Tên gọi của thuật toán này được đặt theo tên của ba nhà phát minh. Mã hóa RSA có khả năng được sử dụng trong nhiều hệ thống khác nhau nhằm bảo vệ thông tin khi truyền tải qua mạng, cũng như trong việc ký và xác thực chữ ký số" [8]. Nguyên lý hoạt động của RSA dựa trên việc áp dụng cặp khóa, bao gồm khóa công khai (public key) và khóa riêng tư (private key):

- **Khóa công khai (Public key):** Đây là khóa được chia sẻ với mọi người và dùng để mã hóa dữ liệu.
- **Khóa riêng tư (Private key):** Đây là khóa bí mật chỉ được chủ sở hữu biết và dùng để giải mã dữ liệu.

Thực hiện mã hoá và giải mã trong RSA được diễn ra như sau:

- Tạo khóa:
 - Bước 1: Chọn hai số nguyên tố lớn: Chọn hai số nguyên tố lớn và ngẫu nhiên p và q.
 - Bước 2: Tính n và $\phi(n)$: Tính $n = p * q$, và tính $\phi(n) = (p - 1) * (q - 1)$, trong đó $\phi(n)$ là hàm số Euler của n.
 - Bước 3: Chọn số nguyên e: Chọn một số nguyên e sao cho $1 < e < \phi(n)$ và e là số nguyên tố cùng nhau với $\phi(n)$ (không có ước chung nào ngoài 1).
 - Bước 4: Tìm khóa công khai và khóa riêng tư: Khóa công khai là cặp (e, n) , và khóa riêng tư là cặp (d, n) sao cho $d * e \equiv 1 \pmod{\phi(n)}$, nghĩa là d là phần dư của e^{-1} khi chia cho $\phi(n)$.
- Mã hoá và giải mã:
 - Mã hoá (Encryption): Người gửi sử dụng khóa công khai (e, n) để mã hoá thông điệp M thành ciphertext C, theo công thức: $C = M^e \pmod{n}$.
 - Giải mã (Decryption): Người nhận sử dụng khóa riêng tư (d, n) để giải mã ciphertext C thành thông điệp M, theo công thức: $M = C^d \pmod{n}$.

3.3.2 Thư viện RSA trong Python

Trong Python, có nhiều thư viện hỗ trợ việc thực hiện mã hóa RSA và các thao tác liên quan đến nó. Một số thư viện phổ biến trong Python để thực hiện RSA bao gồm:

- “**cryptography**”: Đây là thư viện Python mạnh mẽ hỗ trợ nhiều thuật toán mã hóa và bảo mật, trong đó có cả RSA.
- “**PyCryptodome**”: Đây là thư viện mã nguồn mở của Python cung cấp nhiều thuật toán mã hóa và bảo mật, bao gồm RSA.

3.4 ĐƯỜNG CONG ELIPTIC (ECC)

3.4.1 Giới thiệu ECC

Mã hóa đường cong elliptic (Elliptic Curve Cryptography - ECC) là một phương pháp mã hóa ứng dụng các đường cong elliptic trong toán học để xây dựng hệ thống

mã hóa công khai. ECC đóng vai trò quan trọng trong lĩnh vực bảo mật, đặc biệt thích hợp cho các ứng dụng yêu cầu khóa nhỏ và hiệu suất cao.

Ưu điểm của ECC:

- Kích thước khóa nhỏ: ECC có thể đạt được mức độ bảo mật tương đương với RSA với các khóa nhỏ hơn nhiều lần.
- Hiệu suất cao: So với RSA, ECC yêu cầu ít tài nguyên tính toán hơn, giúp tăng hiệu suất trong việc mã hóa và giải mã.

3.4.2 Mật mã ECC

Mật mã ECC là một kỹ thuật mã hóa khóa công khai dựa trên lý thuyết đường cong elliptic có thể được sử dụng để tạo các khóa mật mã nhanh hơn, nhỏ hơn và hiệu quả hơn. ECC tạo khóa thông qua các thuộc tính của phương trình đường cong elliptic thay vì phương pháp tạo truyền thống như là tích của các số nguyên tố rất lớn. Quá trình hoạt động của ECC bao gồm việc tạo ra cặp khóa công khai và khóa riêng tư, mã hóa thông điệp và giải mã như sau:

- Tạo khóa:
 - Bước 1: Chọn đường cong elliptic: Chọn một đường cong elliptic, thường được biểu diễn trong hệ tọa độ affine hoặc các hệ tọa độ khác.
 - Bước 2: Tạo cặp khóa:
 - Khóa riêng tư (Private Key): Chọn một số nguyên ngẫu nhiên làm khóa riêng tư.
 - Khóa công khai (Public Key): Tính điểm trên đường cong elliptic bằng cách nhân khóa riêng tư với điểm gốc trên đường cong, tạo ra khóa công khai.
- Mã hóa:
 - Bước 1: Lựa chọn thông điệp: Người gửi chọn thông điệp cần mã hóa.
 - Bước 2: Lựa chọn điểm ngẫu nhiên: Người gửi chọn một điểm ngẫu nhiên trên đường cong elliptic.
 - Bước 3: Tính toán:

100

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Người gửi thực hiện phép nhân điểm ngẫu nhiên với khóa công khai của người nhận.
- Tính toán phép cộng điểm giữa điểm ngẫu nhiên và điểm đã được nhân với khóa công khai.
- Mã hoá thông điệp bằng cách kết hợp thông điệp với các toạ độ x của điểm ngẫu nhiên và điểm cộng đã tính được.

- Giải mã:

- Bước 1: Nhận thông điệp: Người nhận nhận được thông điệp đã được mã hoá.
- Bước 2: Tính toán:
 - Người nhận sử dụng khóa riêng tư của mình để giải mã các thành phần trong thông điệp mã hoá, bằng cách sử dụng các toạ độ x được trích xuất từ thông điệp.
 - Tính toán phép cộng điểm giữa điểm đã giải mã và điểm ngẫu nhiên ban đầu đã được gửi đi.
 - Giải mã thông điệp từ thông điệp được trích xuất từ phép cộng điểm này.

3.4.3 So sánh ECC và RSA

So sánh độ an toàn giữa ECC và RSA thường liên quan đến độ dài của các khóa cần thiết để đảm bảo mức độ an toàn tương đương.

- Độ dài khóa:

- ECC: Để đạt được mức độ an toàn tương đương, ECC yêu cầu khóa ngắn hơn so với RSA.
- RSA: Đối với RSA, độ dài khóa cần phải lớn hơn để đạt được cùng mức độ an toàn so với ECC.

- Hiệu suất:

- ECC: ECC yêu cầu ít tài nguyên tính toán hơn so với RSA khi sử dụng cùng mức độ an toàn.

- RSA: RSA cần nhiều tài nguyên tính toán hơn đối với cùng mức độ an toàn so với ECC, do đó có thể có hiệu suất kém hơn trong một số trường hợp.
- Bảo mật:
 - ECC: ECC dựa trên độ khó của việc giải quyết bài toán điểm trên đường cong elliptic, một bài toán rất khó khiến cho việc tìm ra khóa riêng tư từ khóa công khai trở nên khó khăn.
 - RSA: RSA dựa trên độ khó của việc phân tích thành phần nguyên tố của một số nguyên lớn hợp thành từ hai số nguyên tố ngẫu nhiên. Đối với RSA, việc tìm ra khóa riêng tư từ khóa công khai cũng là một bài toán khó.

3.5 BÀI TẬP THỰC HÀNH

3.5.1 Bài thực hành 01: Tạo ứng dụng mã hoá Caesar

Tạo ứng dụng desktop demo cho Mật mã Caesar.

Hướng dẫn:

- Clone Git repo về máy tính thực hành (Mở trong CMD):

```
git clone <đường dẫn git repo của sinh viên>
cd .\bmmtt-nc-hutech-1511060249\
code .
```

- Tại giao diện VS Code, mở Terminal, pull mới code về:

```
git pull origin main
```

```
PS C:\Users\PHUOCNTMH\bmmtt-nc-hutech-1511060249> git pull origin main
From https://github.com/mrphuoc020597/bmmtt-nc-hutech-1511060249
 * branch            main       -> FETCH_HEAD
Already up to date.
PS C:\Users\PHUOCNTMH\bmmtt-nc-hutech-1511060249>
```

- Tách nhánh lab03 từ nhánh main.

```
git checkout -b lab03
```

102

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Tạo folder “lab-03”. Trong folder “lab-03” tạo file “requirements.txt”.

```
☰ requirements.txt ×
```

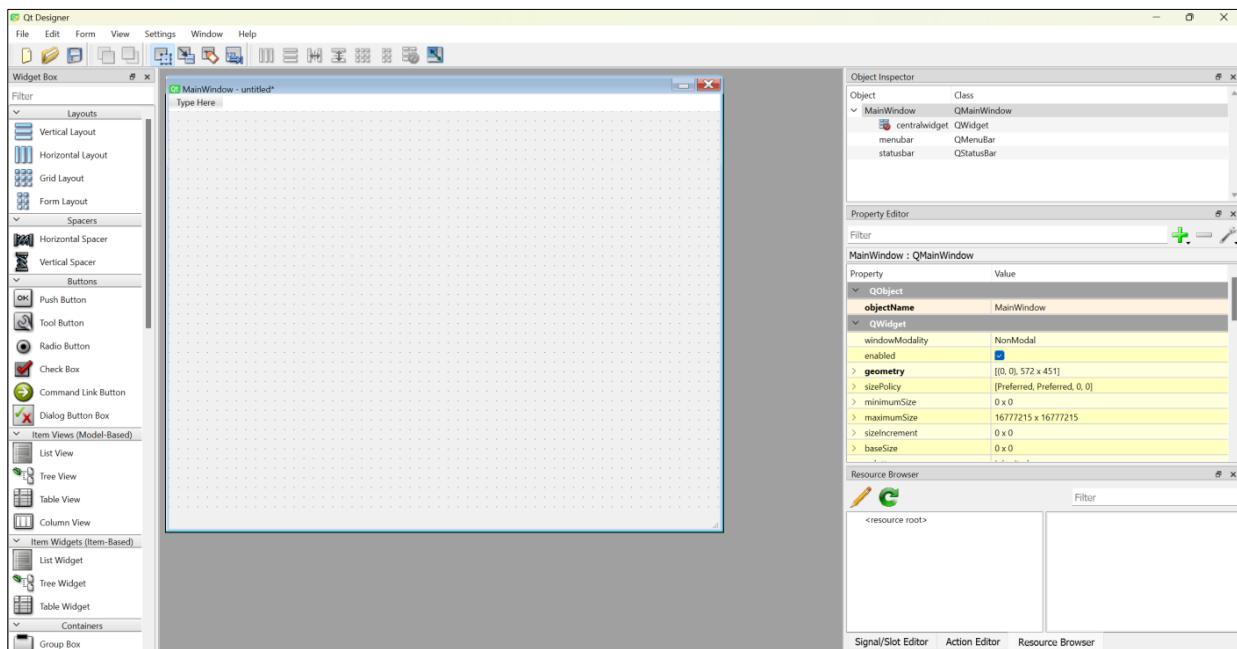
```
lab-03 > ☰ requirements.txt
```

```
1 PyQt5  
2 requests
```

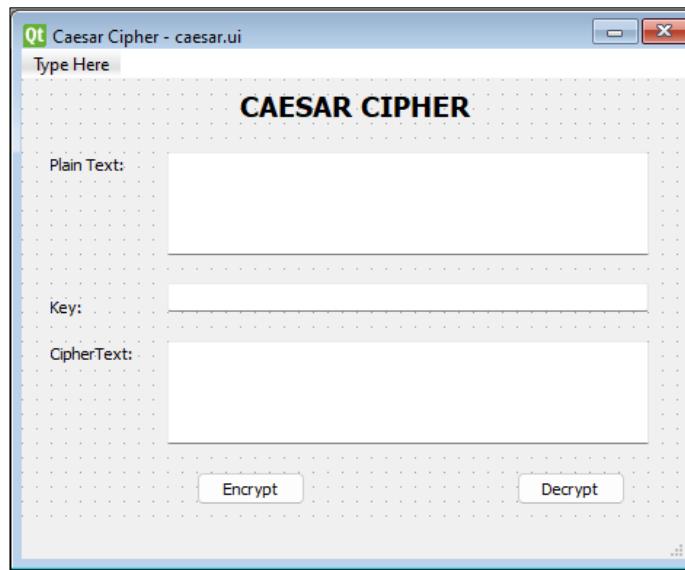
- Vào Terminal, chạy các lệnh sau:

```
cd lab-03  
pip install -r .\requirements.txt
```

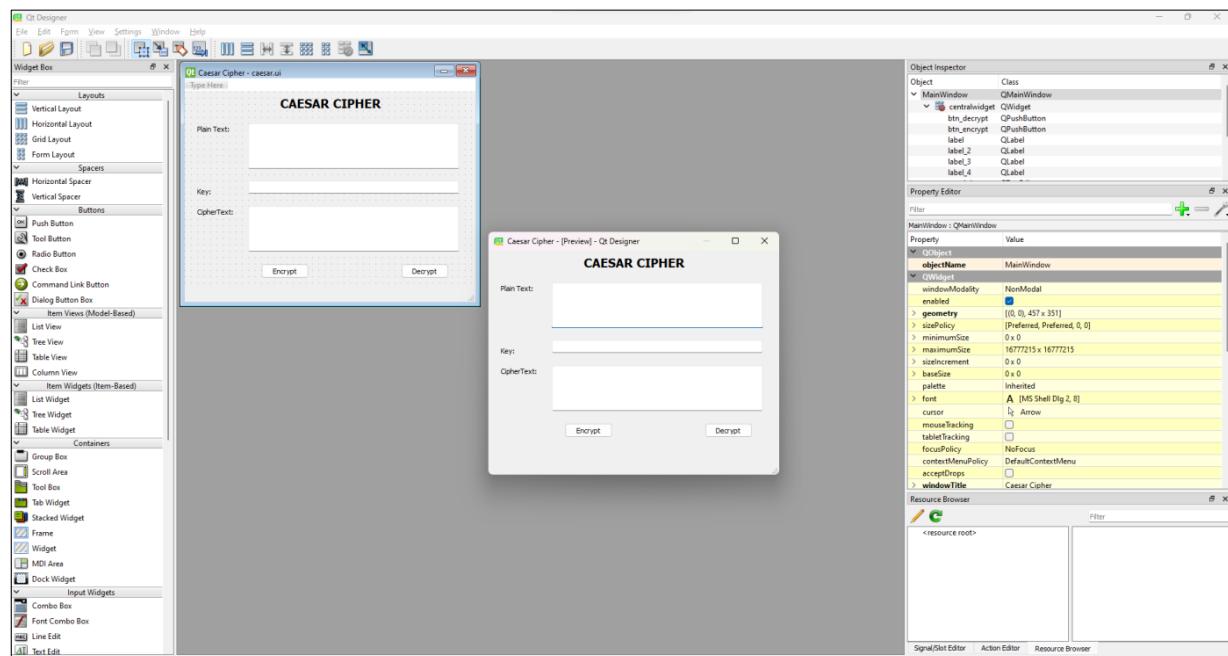
- Trong folder “lab-03” tạo folder “ui”, “platforms”.
- Tải về và cài đặt công cụ QtDesigner tại địa chỉ: <https://build-system.fman.io/qt-designer-download>. Sau khi cài đặt, mở phần mềm QtDesigner.
- Ở phần chọn template, chúng ta chọn “Main Window” ➔ Nhấn “Create”.
- Giao diện của QtDesigner xuất hiện. Bên trái là các Widget, ở giữa là giao diện xem trước và bên phải là tùy chỉnh các thuộc tính Property.



- Tiến hành sử dụng các Widget và thiết kế giao diện cho ứng dụng như sau:



- Sau khi thiết kế xong, nhấn tổ hợp phím Ctrl + R để xem lại giao diện khi thực thi.



- Tiến hành lưu file thiết kế .ui vào folder "ui" trong "lab-03" với tên "caesar.ui".
- Chuyển giao diện sang code PyQt5 bằng cách chạy lệnh sau trong Terminal của VS Code (thực thi tại thư mục "lab-03"):

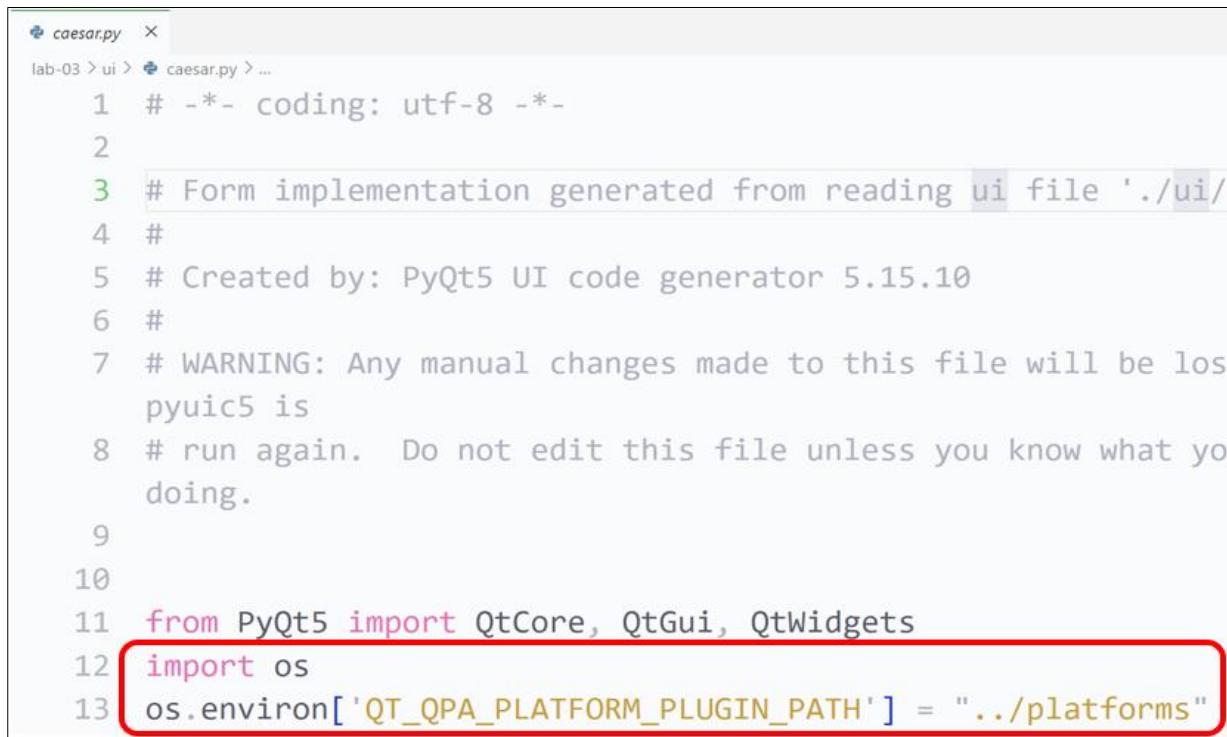
```
pyuic5 -x ./ui/caesar.ui -o ./ui/caesar.py
```

104

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Copy các file từ thư mục sau (<đường_dẫn_thư_mục_cài_Python>\Lib\site-packages\PyQt5\Qt5\plugins\platforms) sang thư mục “platforms” của project.
- Thêm hai dòng sau vào đầu file “caesar.py”:

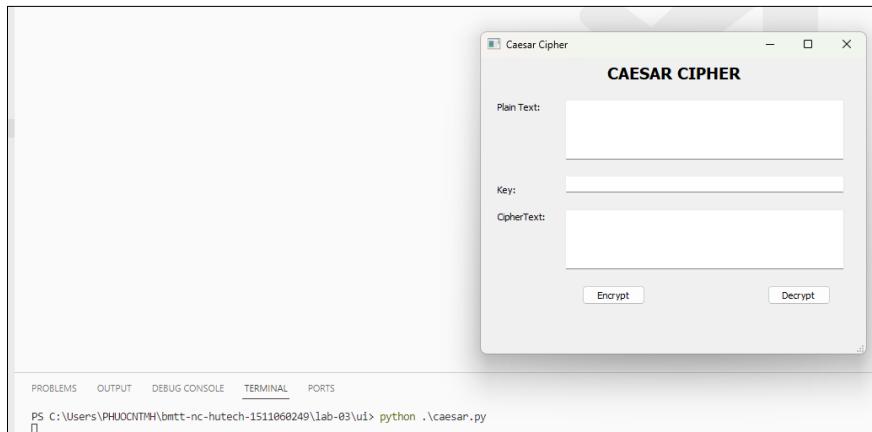
```
import os
os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = "../platforms"
```



```
caesar.py  x
lab-03 > ui > caesar.py > ...
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file './ui/
4 #
5 # Created by: PyQt5 UI code generator 5.15.10
6 #
7 # WARNING: Any manual changes made to this file will be lost if
8 # pyuic5 is
9 # run again. Do not edit this file unless you know what you're
10 # doing.
11
12
13 from PyQt5 import QtCore, QtGui, QtWidgets
14 import os
15 os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = "../platforms"
```

- Kiểm tra thực thi của giao diện:

```
python .\caesar.py
```



- Vừa rồi, chúng ta đã làm xong front-end, tiếp theo, chúng ta sẽ làm phần back-end.
- Trong folder “lab-03” tạo file “caesar_cipher.py” (Lưu ý: Kiểm tra tên của các button cho đúng).

```

caesar_cipher.py X

lab-03 > caesar_cipher.py > MyApp > call_api_decrypt

1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
3 from ui.caesar import Ui_MainWindow
4 import requests
5
6 class MyApp(QMainWindow):
7     def __init__(self):
8         super().__init__()
9         self.ui = Ui_MainWindow()
10        self.ui.setupUi(self)
11        self.ui.btn_encrypt.clicked.connect(self.call_api_encrypt)
12        self.ui.btn_decrypt.clicked.connect(self.call_api_decrypt)
13
14    def call_api_encrypt(self):
15        url = "http://127.0.0.1:5000/api/caesar/encrypt"
16        payload = {
17            "plain_text": self.ui.txt_plain_text.toPlainText(),
18            "key": self.ui.txt_key.text()
19        }
20        try:
21            response = requests.post(url, json=payload)
22            if response.status_code == 200:
23                data = response.json()
24                self.ui.txt_cipher_text.setText(data["encrypted_message"])
25
26                msg = QMessageBox()
27                msg.setIcon(QMessageBox.Information)
28                msg.setText("Encrypted Successfully")
29                msg.exec_()
30            else:
31                print("Error while calling API")
32        except requests.exceptions.RequestException as e:
33            print("Error: %s" % e.message)

```

```

34
35    def call_api_decrypt(self):
36        url = "http://127.0.0.1:5000/api/caesar/decrypt"
37        payload = {
38            "cipher_text": self.ui.txt_cipher_text.toPlainText(),
39            "key": self.ui.txt_key.text()
40        }

```

106

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

```
41     try:
42         response = requests.post(url, json=payload)
43         if response.status_code == 200:
44             data = response.json()
45             self.ui.txt_plain_text.setText(data["decrypted_message"])
46
47         msg = QMessageBox()
48         msg.setIcon(QMessageBox.Information)
49         msg.setText("Decrypted Successfully")
50         msg.exec_()
51     else:
52         print("Error while calling API")
53     except requests.exceptions.RequestException as e:
54         print("Error: %s" % e.message)
55
56 if __name__ == "__main__":
57     app = QApplication(sys.argv)
58     window = MyApp()
59     window.show()
60     sys.exit(app.exec_())
```

- Tiến hành kiểm tra ứng dụng:
 - Bước 1: Split Terminal của VS Code, di chuyển vào folder “lab-02” và chạy file “api.py”.

```
python .\api.py
```

- Bước 2: Tại Terminal của “lab-03”. Tiến hành chạy file “caesar_cipher.py”.

```
python .\caesar_cipher.py
```

- Bước 3: Giao diện ứng dụng xuất hiện, tiến hành kiểm tra các chức năng mã hoá, giải mã.
- Lưu ý: Cần phải chạy lại project của “lab-02” vì chương trình sẽ kết nối và sử dụng API encrypt và decrypt của Mã hoá Caesar đã làm ở tuần trước. Khi click button sẽ gửi một HTTP Request (hai project đóng vai trò client-server).

```

PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249> cd lab-02
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-02> python .\api.py
* Serving Flask app 'api'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
I server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.16.6.9:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 929-074-328
127.0.0.1 - - [27/Dec/2023 15:57:09] "POST /api/caesar/encrypt HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 15:57:35] "POST /api/caesar/decrypt HTTP/1.1" 200 -

```

- Kết quả kiểm tra:

The Caesar Cipher application window shows "Plain Text: HUTECH", "Key: 4", and "CipherText: LYXIGL". A modal dialog says "Encrypted Successfully".

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-03> cd ..
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-03> python .\caesar_cipher.py
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249> cd lab-02
PS C:\Users\PHUOCNTMH\bmtt-nc-hutech-1511060249\lab-02> python .\api.py
* Serving Flask app 'api'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
I server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.16.6.9:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 929-074-328
127.0.0.1 - - [27/Dec/2023 15:57:09] "POST /api/caesar/encrypt HTTP/1.1" 200 -
127.0.0.1 - - [27/Dec/2023 15:57:35] "POST /api/caesar/decrypt HTTP/1.1" 200 -

```

- Commit code:

```

git add .
git commit -m "[add] lab-03 ui caesar"

```

3.5.2 Bài thực hành 02: Tạo ứng dụng mã hóa RSA

Tạo ứng dụng desktop demo cho Mật mã RSA. Hướng dẫn:

- Trong folder "lab-03" tạo folder "cipher". Trong folder "cipher" tạo folder "rsa".
- Trong folder "rsa" tạo folder "keys" để chứa private key và public key.

108

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Cập nhật file “requirements.txt”, thêm nội dung sau vào:

```
Flask>=2.3.2
rsa>=4.9
```

- Cài đặt các gói cần thiết:

```
pip install -r .\requirements.txt
```

- Tạo file “rsa_cipher.py” trong folder “rsa”.

```
rsa_cipher.py ×

lab-03 > rsa_cipher.py > MyApp > call_api_encrypt
1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
3 from ui.rsa import Ui_MainWindow
4 import requests
5
6 class MyApp(QMainWindow):
7     def __init__(self):
8         super().__init__()
9         self.ui = Ui_MainWindow()
10        self.ui.setupUi(self)
11        self.ui.btn_gen_keys.clicked.connect(self.call_api_gen_keys)
12        self.ui.btn_encrypt.clicked.connect(self.call_api_encrypt)
13        self.ui.btn_decrypt.clicked.connect(self.call_api_decrypt)
14        self.ui.btn_sign.clicked.connect(self.call_api_sign)
15        self.ui.btn_verify.clicked.connect(self.call_api_verify)
16
17    def call_api_gen_keys(self):
18        url = "http://127.0.0.1:5000/api/rsa/generate_keys"
19        try:
20            response = requests.get(url)
21            if response.status_code == 200:
22                data = response.json()
23                msg = QMessageBox()
24                msg.setIcon(QMessageBox.Information)
25                msg.setText(data["message"])
26                msg.exec_()
27            else:
28                print("Error while calling API")
29        except requests.exceptions.RequestException as e:
30            print("Error: %s" % e.message)
```

```
32  def call_api_encrypt(self):
33      url = "http://127.0.0.1:5000/api/rsa/encrypt"
34      payload = {
35          "message": self.ui.txt_plain_text.toPlainText(),
36          "key_type": "public"
37      }
38      try:
39          response = requests.post(url, json=payload)
40          if response.status_code == 200:
41              data = response.json()
42              self.ui.txt_cipher_text.setText(data["encrypted_message"])
43
44          msg = QMessageBox()
45          msg.setIcon(QMessageBox.Information)
46          msg.setText("Encrypted Successfully")
47          msg.exec_()
48      else:
49          print("Error while calling API")
50  except requests.exceptions.RequestException as e:
51      print("Error: %s" % e.message)
52
```

```
53  def call_api_decrypt(self):
54      url = "http://127.0.0.1:5000/api/rsa/decrypt"
55      payload = {
56          "ciphertext": self.ui.txt_cipher_text.toPlainText(),
57          "key_type": "private"
58      }
59      try:
60          response = requests.post(url, json=payload)
61          if response.status_code == 200:
62              data = response.json()
63              self.ui.txt_plain_text.setText(data["decrypted_message"])
64
65          msg = QMessageBox()
66          msg.setIcon(QMessageBox.Information)
67          msg.setText("Decrypted Successfully")
68          msg.exec_()
69      else:
70          print("Error while calling API")
71  except requests.exceptions.RequestException as e:
72      print("Error: %s" % e.message)
73
```

```
74  def call_api_sign(self):
75      url = "http://127.0.0.1:5000/api/rsa/sign"
76      payload = {
77          "message": self.ui.txt_info.toPlainText(),
78      }
```

110

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

```
79     try:
80         response = requests.post(url, json=payload)
81         if response.status_code == 200:
82             data = response.json()
83             self.ui.txt_sign.setText(data["signature"])
84
85             msg = QMessageBox()
86             msg.setIcon(QMessageBox.Information)
87             msg.setText("Signed Successfully")
88             msg.exec_()
89         else:
90             print("Error while calling API")
91     except requests.exceptions.RequestException as e:
92         print("Error: %s" % e.message)
93
```

```
94     def call_api_verify(self):
95         url = "http://127.0.0.1:5000/api/rsa/verify"
96         payload = {
97             "message": self.ui.txt_info.toPlainText(),
98             "signature": self.ui.txt_sign.toPlainText()
99         }
100        try:
101            response = requests.post(url, json=payload)
102            if response.status_code == 200:
103                data = response.json()
104                if (data["is_verified"]):
105                    msg = QMessageBox()
106                    msg.setIcon(QMessageBox.Information)
107                    msg.setText("Verified Successfully")
108                    msg.exec_()
109                else:
110                    msg = QMessageBox()
111                    msg.setIcon(QMessageBox.Information)
112                    msg.setText("Verified Fail")
113                    msg.exec_()
114            else:
115                print("Error while calling API")
116        except requests.exceptions.RequestException as e:
117            print("Error: %s" % e.message)
118
119    if __name__ == "__main__":
120        app = QApplication(sys.argv)
121        window = MyApp()
122        window.show()
123        sys.exit(app.exec_())
```

- Tạo file “`__init__.py`” trong folder “rsa”.

```
pu __init__.py X
lab-03 > cipher > rsa > pu __init__.py
1 from .rsa_cipher import RSACipher
```

- Tạo file “api.py” trong thư mục “lab-03”.

```
pu api.py 1 X
lab-03 > pu api.py > rsa_decrypt
1 from flask import Flask, request, jsonify
2 from cipher.rsa import RSACipher
3
4
5 app = Flask(__name__)
6
7 #RSA CIPHER ALGORITHM
8 rsa_cipher = RSACipher()
9
10 @app.route('/api/rsa/generate_keys', methods=['GET'])
11 def rsa_generate_keys():
12     rsa_cipher.generate_keys()
13     return jsonify({'message': 'Keys generated successfully'})
14
15 @app.route("/api/rsa/encrypt", methods=["POST"])
16 def rsa_encrypt():
17     data = request.json
18     message = data['message']
19     key_type = data['key_type']
20     private_key, public_key = rsa_cipher.load_keys()
21     if key_type == 'public':
22         key = public_key
23     elif key_type == 'private':
24         key = private_key
25     else:
26         return jsonify({'error': 'Invalid key type'})
27     encrypted_message = rsa_cipher.encrypt(message, key)
28     encrypted_hex = encrypted_message.hex()
29     return jsonify({'encrypted_message': encrypted_hex})
30
```

```
31 @app.route("/api/rsa/decrypt", methods=["POST"])
32 def rsa_decrypt():
33     data = request.json
34     ciphertext_hex = data['ciphertext']
35     key_type = data['key_type']
36     private_key, public_key = rsa_cipher.load_keys()
37     if key_type == 'public':
```

112

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

```
38     key = public_key
39 elif key_type == 'private':
40     key = private_key
41 else:
42     return jsonify({'error': 'Invalid key type'})
43 ciphertext = bytes.fromhex(ciphertext_hex)
44 decrypted_message = rsa_cipher.decrypt(ciphertext, key)
45 return jsonify({'decrypted_message': decrypted_message})
46
47 @app.route('/api/rsa/sign', methods=['POST'])
48 def rsa_sign_message():
49     data = request.json
50     message = data['message']
51     private_key, _ = rsa_cipher.load_keys()
52     signature = rsa_cipher.sign(message, private_key)
53     signature_hex = signature.hex()
54     return jsonify({'signature': signature_hex})
55
56 @app.route('/api/rsa/verify', methods=['POST'])
57 def rsa_verify_signature():
58     data = request.json
59     message = data['message']
60     signature_hex = data['signature']
61     public_key, _ = rsa_cipher.load_keys()
62     signature = bytes.fromhex(signature_hex)
63     is_verified = rsa_cipher.verify(message, signature, public_key)
64     return jsonify({'is_verified': is_verified})
65 #main function
66 if __name__ == "__main__":
67     app.run(host="0.0.0.0", port=5000, debug=True)
```

- Khởi động development server.

```
python .\api.py
```

- Khởi động Postman, tạo mới collection "RSA" có 04 HTTP Request là "generate_keys", "encrypt", "decrypt", "sign", "verify":

```
1 curl --location 'http://127.0.0.1:5000/api/rsa/generate_keys'
```

```
1 curl --location 'http://127.0.0.1:5000/api/rsa/encrypt' \
2 --header 'Content-Type: application/json' \
3 --data '{
4     "message": "",
5     "key_type": "public"
6 }'
```

```

1 curl --location 'http://127.0.0.1:5000/api/rsa/decrypt' \
2 --header 'Content-Type: application/json' \
3 --data '{
4     "ciphertext": "",
5     "key_type": "private"
6 }'

```

```

1 curl --location 'http://127.0.0.1:5000/api/rsa/sign' \
2 --header 'Content-Type: application/json' \
3 --data '{
4     "message": "HUTECH University"
5 }'

```

```

1 curl --location 'http://127.0.0.1:5000/api/rsa/verify' \
2 --header 'Content-Type: application/json' \
3 --data '{
4     "message": "HUTECH University",
5     "signature": ""
6 }'

```

- Tiến hành bấm nút “Send” để kiểm tra các API.
- Kết quả tạo key thành công (kiểm tra folder “key” thấy có 2 key là publicKey.pem và privateKey.pem):

The screenshot shows the Postman application interface. On the left, there's a request builder for a GET method to the endpoint {{base_url}}/rsa/generate_keys. The 'Body' tab is selected, showing the JSON response: { "message": "Keys generated successfully"}. On the right, a 'Code snippet' panel displays the curl command used for this request: curl --location 'http://127.0.0.1:5000/api/rsa/generate_keys'. Below the code snippet, the response status is shown as 200 OK with a size of 212 B.

- Kết quả mã hoá:

114

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

The screenshot shows the Postman interface for the RSA/encrypt API. The request method is POST to `((base_url))/rsa/encrypt`. The body contains the following JSON:

```
1 {
2   ... "message": "Nguyen Trong Minh Hong Phuoc",
3   ... "key_type": "public"
4 }
```

The response status is 200 OK with a size of 452 B. The response body is:

```
1 {
2   "encrypted_message":
3     "53d98858ada3cf2a6b63ebdb19d5b87b5ba9c66d175c056503c01409cb8dc5764f37b7b85650c21982824722fc6d2e1669c2f5684
4       efc07f997055fead16f2a18098b105f749ae65034fa32f471ec5b737b4755a639cf1cbb7d15a5bb5eff05c53950c09d0ce21f0cf3a0
5         13e3cb58703793f0b8482679fc19cd121b15297"
```

- Kết quả giải mã:

The screenshot shows the Postman interface for the RSA/decrypt API. The request method is POST to `((base_url))/rsa/decrypt`. The body contains the following JSON:

```
1 {
2   ... "ciphertext":
3     "bfef50e13b2a4e5e9c1dcb904da2d4b259acf07e613b59bcd9b1058c702a1fc2f59d8d69ac90e90c4986ab1d31b2b1fe93302ae0e7cd
4       decfea262839156d2faacf0203f642b5c97f0288600771df4ac23e4a62e2fc4c4649742b696da65c02328335f3de91156d9023abde7448
5         9fb36fb059360d49e990ddbd6df7fafdf6e696",
6   ... "key_type": "private"
7 }
```

The response status is 200 OK with a size of 223 B. The response body is:

```
1 {
2   "decrypted_message": "Nguyen Trong Minh Hong Phuoc"
3 }
```

- Kết quả tạo chữ ký:

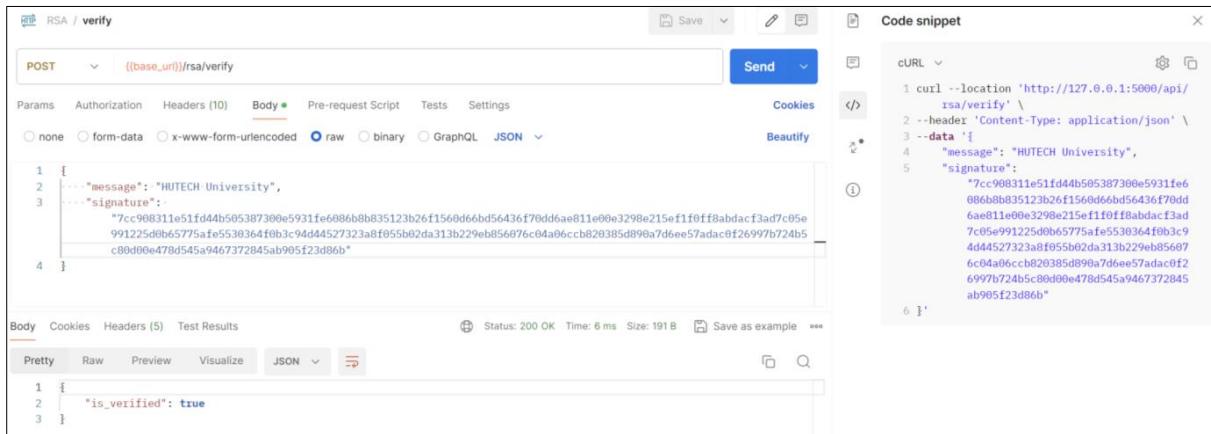
The screenshot shows the Postman interface for the RSA/sign API. The request method is POST to `((base_url))/rsa/sign`. The body contains the following JSON:

```
1 {
2   ... "message": "HUTECH University"
3 }
```

The response status is 200 OK with a size of 444 B. The response body is:

```
1 {
2   "signature":
3     "7cc908311e51fd44b505387300e5931fe6086b80835123b26f1560d66bd56436f70ddae811e00e3298e215ef1ff0ff8abdacf3adac7c05e
4       99125df0b65775afe5530364fb3c94d4527323a8f055b02da313b229eb856076c94a06ccb820385d890a7d6ee57adac0f26997b724b5
5         c80d00e478d545a9467372845ab905f23d86b"
```

- Kết quả xác thực chữ ký:



The screenshot shows the Postman interface with a POST request to `((base_url))/rsa/verify`. The request body is raw JSON:

```

1 {
2   ... "message": "HUTECH University",
3   ... "signature": "7cc908311e51fd44b505387380e5931fe6086b8b835123b26f1560d66bd56436f70ddae6a811e00e3298e215ef110ff8abdcf3ad7c05e99125d0b65775afe5530364fb3c94dd4527323a8f055b02da313b229eb856676c0a06ccb20385d890a7d6ee57adac0f26997b724b5c80d00e478545a9467372845ab905f23d56b"
4 }

```

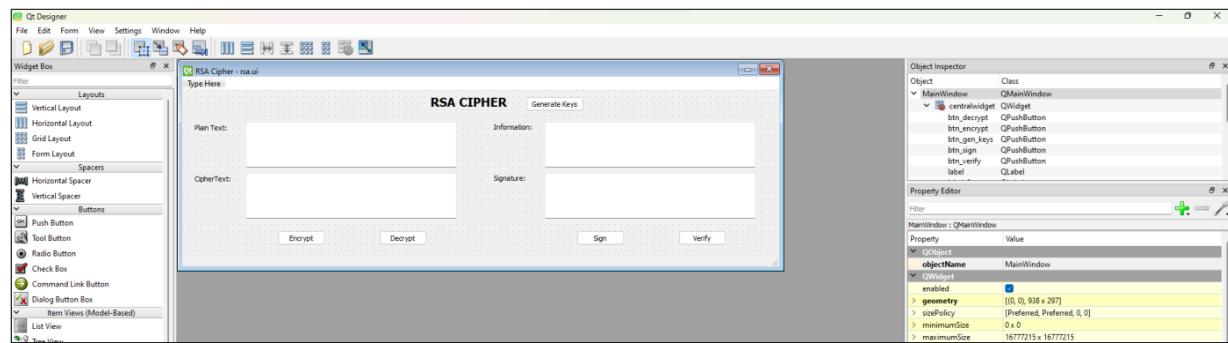
The response status is 200 OK with a size of 191 B. The JSON response body is:

```

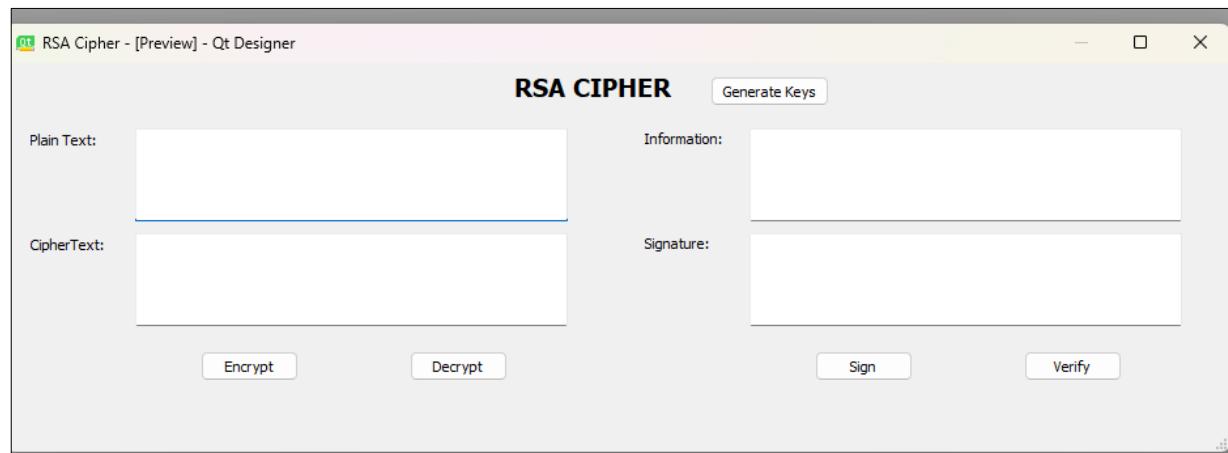
1 {
2   "is_verified": true
3 }

```

- Mở QtDesigner và thiết kế giao diện cho ứng dụng:



- Sau khi thiết kế xong, nhấn tổ hợp phím Ctrl + R để xem lại giao diện khi thực thi.



- Tiến hành lưu file thiết kế .ui vào folder "ui" trong "lab-03" với tên "rsa.ui".

116

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Chuyển giao diện sang code PyQt5 bằng cách chạy lệnh sau trong Terminal của VS Code (thực thi tại thư mục “lab-03”):

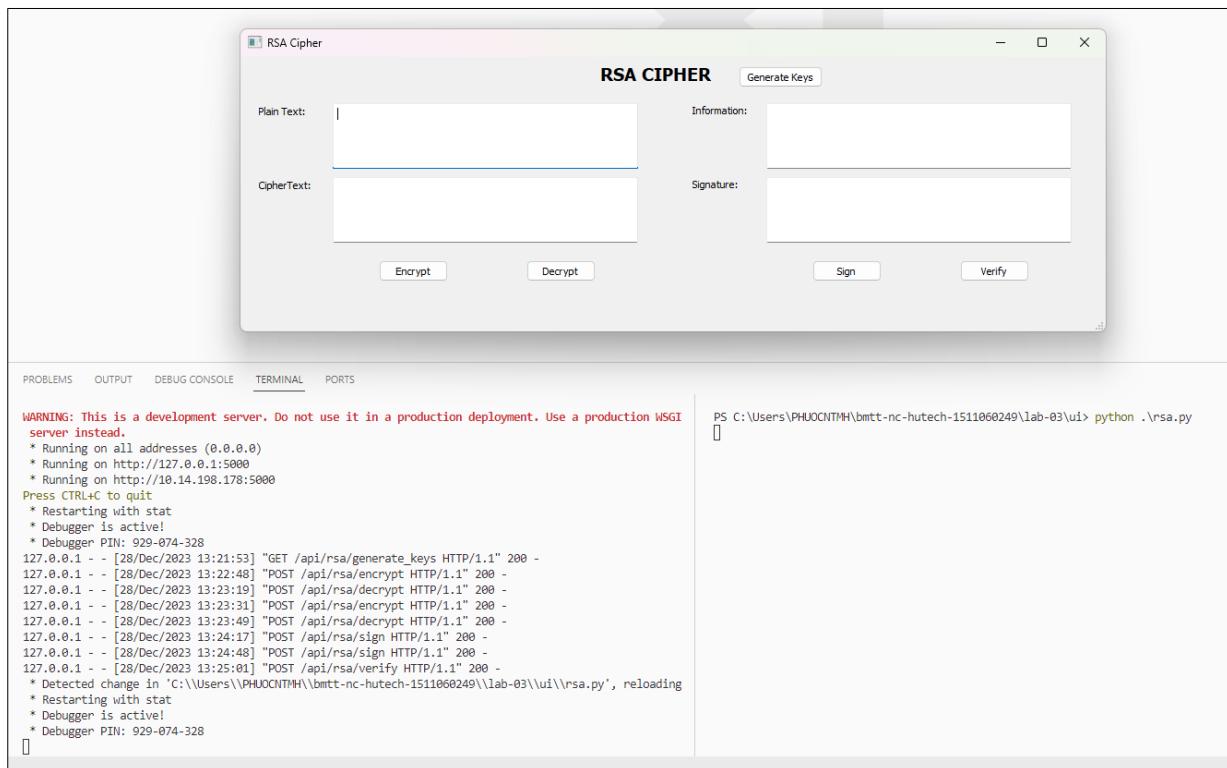
```
pyuic5 -x ./ui/rsa.ui -o ./ui/rsa.py
```

- Thêm hai dòng sau vào đầu file “rsa.py”:

```
import os  
os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = "../platforms"
```

- Kiểm tra thực thi của giao diện:

```
python .\rsa.py
```



- Vừa rồi, chúng ta đã làm xong front-end, tiếp theo, chúng ta sẽ làm phần back-end.
- Trong folder “lab-03” tạo file “rsa_cipher.py” (Lưu ý: Kiểm tra tên của các button cho đúng).

```
rsa_cipher.py X
lab-03 > rsa_cipher.py > MyApp > call_api_encrypt
1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
3 from ui.rsa import Ui_MainWindow
4 import requests
5
6 class MyApp(QMainWindow):
7     def __init__(self):
8         super().__init__()
9         self.ui = Ui_MainWindow()
10        self.ui.setupUi(self)
11        self.ui.btn_gen_keys.clicked.connect(self.call_api_gen_keys)
12        self.ui.btn_encrypt.clicked.connect(self.call_api_encrypt)
13        self.ui.btn_decrypt.clicked.connect(self.call_api_decrypt)
14        self.ui.btn_sign.clicked.connect(self.call_api_sign)
15        self.ui.btn_verify.clicked.connect(self.call_api_verify)
16
17    def call_api_gen_keys(self):
18        url = "http://127.0.0.1:5000/api/rsa/generate_keys"
19        try:
20            response = requests.get(url)
21            if response.status_code == 200:
22                data = response.json()
23                msg = QMessageBox()
24                msg.setIcon(QMessageBox.Information)
25                msg.setText(data["message"])
26                msg.exec_()
27            else:
28                print("Error while calling API")
29        except requests.exceptions.RequestException as e:
30            print("Error: %s" % e.message)
31
```

```
32    def call_api_encrypt(self):
33        url = "http://127.0.0.1:5000/api/rsa/encrypt"
34        payload = {
35            "message": self.ui.txt_plain_text.toPlainText(),
36            "key_type": "public"
37        }
38        try:
39            response = requests.post(url, json=payload)
40            if response.status_code == 200:
41                data = response.json()
42                self.ui.txt_cipher_text.setText(data["encrypted_message"])
43
```

```

44     msg = QMessageBox()
45     msg.setIcon(QMessageBox.Information)
46     msg.setText("Encrypted Successfully")
47     msg.exec_()
48 else:
49     print("Error while calling API")
50 except requests.exceptions.RequestException as e:
51     print("Error: %s" % e.message)
52

```

```

53     def call_api_decrypt(self):
54         url = "http://127.0.0.1:5000/api/rsa/decrypt"
55         payload = {
56             "ciphertext": self.ui.txt_cipher_text.toPlainText(),
57             "key_type": "private"
58         }
59         try:
60             response = requests.post(url, json=payload)
61             if response.status_code == 200:
62                 data = response.json()
63                 self.ui.txt_plain_text.setText(data["decrypted_message"])
64
65             msg = QMessageBox()
66             msg.setIcon(QMessageBox.Information)
67             msg.setText("Decrypted Successfully")
68             msg.exec_()
69         else:
70             print("Error while calling API")
71         except requests.exceptions.RequestException as e:
72             print("Error: %s" % e.message)
73

```

```

74     def call_api_sign(self):
75         url = "http://127.0.0.1:5000/api/rsa/sign"
76         payload = {
77             "message": self.ui.txt_info.toPlainText(),
78         }
79         try:
80             response = requests.post(url, json=payload)
81             if response.status_code == 200:
82                 data = response.json()
83                 self.ui.txt_sign.setText(data["signature"])
84
85             msg = QMessageBox()
86             msg.setIcon(QMessageBox.Information)
87             msg.setText("Signed Successfully")
88             msg.exec_()
89         else:

```

```

90         print("Error while calling API")
91     except requests.exceptions.RequestException as e:
92         print("Error: %s" % e.message)

```

```

93
94     def call_api_verify(self):
95         url = "http://127.0.0.1:5000/api/rsa/verify"
96         payload = {
97             "message": self.ui.txt_info.toPlainText(),
98             "signature": self.ui.txt_sign.toPlainText()
99         }
100    try:
101        response = requests.post(url, json=payload)
102        if response.status_code == 200:
103            data = response.json()
104            if (data["is_verified"]):
105                msg = QMessageBox()
106                msg.setIcon(QMessageBox.Information)
107                msg.setText("Verified Successfully")
108                msg.exec_()
109            else:
110                msg = QMessageBox()
111                msg.setIcon(QMessageBox.Information)
112                msg.setText("Verified Fail")
113                msg.exec_()
114        else:
115            print("Error while calling API")
116    except requests.exceptions.RequestException as e:
117        print("Error: %s" % e.message)
118
119 if __name__ == "__main__":
120     app = QApplication(sys.argv)
121     window = MyApp()
122     window.show()
123     sys.exit(app.exec_())

```

- Tiến hành kiểm tra ứng dụng:

- Bước 1: Split Terminal của VS Code, di chuyển vào folder “lab-03” và chạy file “api.py”.

```
python .\api.py
```

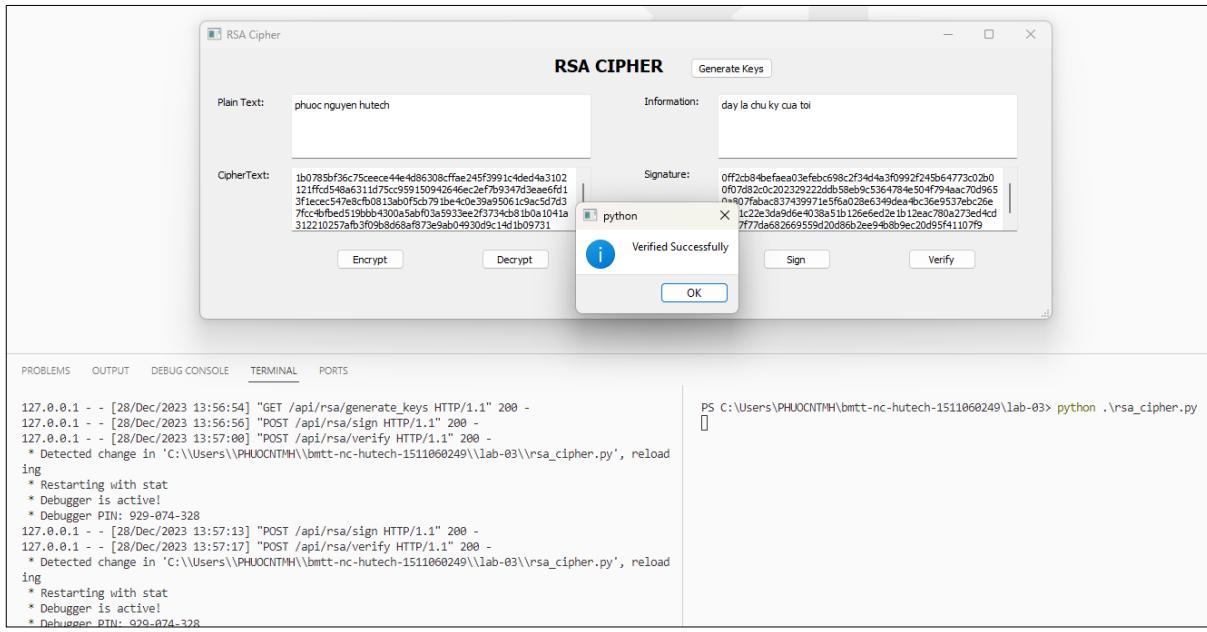
- Bước 2: Tại Terminal của “lab-03”. Tiến hành chạy file “rsa_cipher.py”.

```
python .\rsa_cipher.py
```

120

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

- Bước 3: Giao diện ứng dụng xuất hiện, tiến hành kiểm tra các chức năng tạo khoá, mã hoá, giải mã, ký, xác thực chữ ký.
- Kết quả kiểm tra:



- Commit code:

```
git add .
git commit -m "[add] lab-03 ui rsa"
```

3.5.3 Bài thực hành 03: Tạo ứng dụng mã hoá ECC

Tạo ứng dụng desktop demo cho Mật mã ECC.

Hướng dẫn:

- Trong folder "cipher" tạo folder "ecc".
- Trong folder "ecc" tạo folder "keys" để chứa private key và public key.
- Cập nhật file "requirements.txt", thêm nội dung sau vào:

```
ecdsa
```

- Cài đặt các gói cần thiết:

```
pip install -r .\requirements.txt
```

- Tạo file “ecc_cipher.py” trong folder “ecc”.

```
⌚ ecc_cipher.py ×
lab-03 > cipher > ecc > ⌚ ecc_cipher.py > ...
1 import ecdsa, os
2
3 if not os.path.exists('cipher/ecc/keys'):
4     os.makedirs('cipher/ecc/keys')
5
6 class ECCCipher:
7     def __init__(self):
8         pass
9
10    def generate_keys(self):
11        sk = ecdsa.SigningKey.generate()          # Tạo khóa riêng tư
12        vk = sk.get_verifying_key()              # Lấy khóa công khai từ khóa riêng
13        tư
14
15        with open('cipher/ecc/keys/privateKey.pem', 'wb') as p:
16            p.write(sk.to_pem())
17
18        with open('cipher/ecc/keys/publicKey.pem', 'wb') as p:
19            p.write(vk.to_pem())
20
21    def load_keys(self):
22        with open('cipher/ecc/keys/privateKey.pem', 'rb') as p:
23            sk = ecdsa.SigningKey.from_pem(p.read())
24
25        with open('cipher/ecc/keys/publicKey.pem', 'rb') as p:
26            vk = ecdsa.VerifyingKey.from_pem(p.read())
27
28        return sk, vk
```

```
29    def sign(self, message, key):
30        # Ký dữ liệu bằng khóa riêng tư
31        return key.sign(message.encode('ascii'))
32
33    def verify(self, message, signature, key):
34        _, vk = self.load_keys()
35        try:
36            return vk.verify(signature, message.encode('ascii'))
37        except ecdsa.BadSignatureError:
38            return False
```

- Tạo file “`__init__.py`” trong folder “ecc”.

```
✚ __init__.py ✘
lab-03 > cipher > ecc > ✚ __init__.py
1   from .ecc_cipher import ECCCipher
```

- Cập nhật thêm code vào file “`api.py`” trong thư mục “lab-03”.

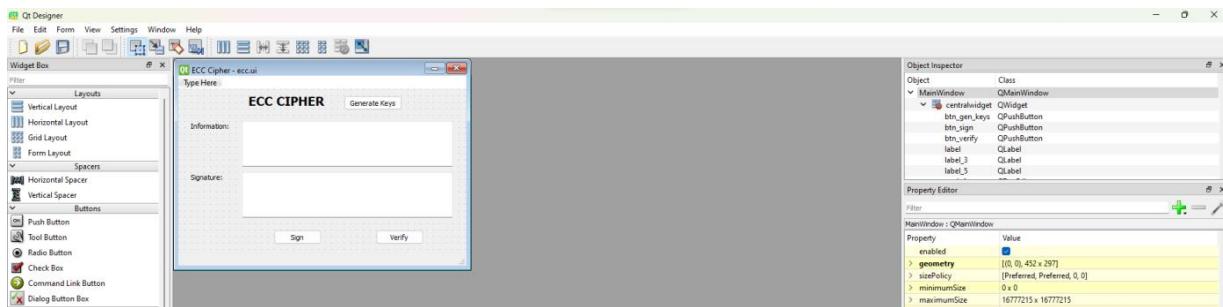
```
✚ api.py ✘
lab-03 > ✚ api.py > ✎ rsa_verify_signature
1   from flask import Flask, request, jsonify
2   from cipher.rsa import RSACipher
3   from cipher.ecc import ECCCipher # Thêm vào đầu file
```

```
66  # Thêm đoạn này trước hàm main
67  #ECC CIPHER ALGORITHM
68  ecc_cipher = ECCCipher()
69
70  @app.route('/api/ecc/generate_keys', methods=['GET'])
71  def ecc_generate_keys():
72      ecc_cipher.generate_keys()
73      return jsonify({'message': 'Keys generated successfully'})
74
75  @app.route('/api/ecc/sign', methods=['POST'])
76  def ecc_sign_message():
77      data = request.json
78      message = data['message']
79      private_key, _ = ecc_cipher.load_keys()
80      signature = ecc_cipher.sign(message, private_key)
81      signature_hex = signature.hex()
82      return jsonify({'signature': signature_hex})
83
84  @app.route('/api/ecc/verify', methods=['POST'])
85  def ecc_verify_signature():
86      data = request.json
87      message = data['message']
88      signature_hex = data['signature']
89      public_key, _ = ecc_cipher.load_keys()
90      signature = bytes.fromhex(signature_hex)
91      is_verified = ecc_cipher.verify(message, signature, public_key)
92      return jsonify({'is_verified': is_verified})
93
```

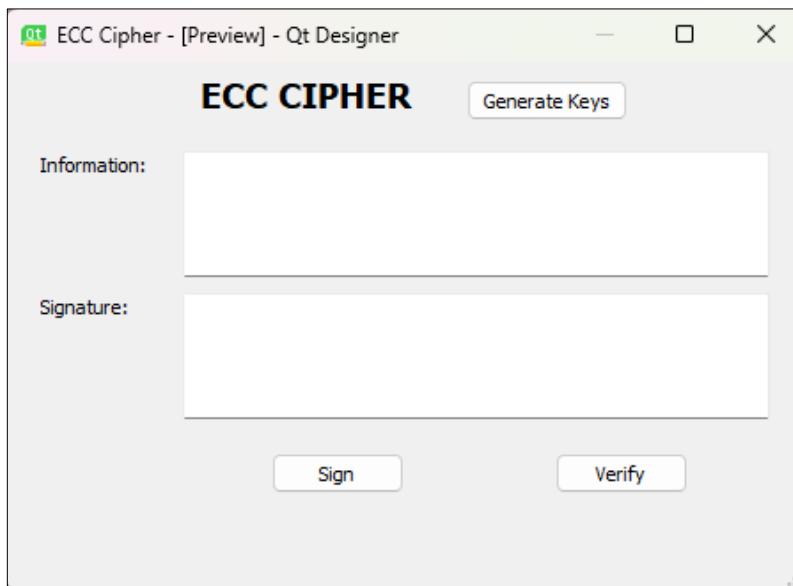
- Khởi động lại development server. Nếu development server đang chạy, nhấn tổ hợp phím `Ctrl + C` để ngắt.

```
python .\api.py
```

- Sinh viên có thể dùng Postman để kiểm tra các API trước khi tiến hành bước tiếp theo.
- Mở QtDesigner và thiết kế giao diện cho ứng dụng:



- Sau khi thiết kế xong, nhấn tổ hợp phím Ctrl + R để xem lại giao diện khi thực thi.



- Tiến hành lưu file thiết kế .ui vào folder "ui" trong "lab-03" với tên "ecc.ui".
- Chuyển giao diện sang code PyQt5 bằng cách chạy lệnh sau trong Terminal của VS Code (thực thi tại thư mục "lab-03"):

```
pyuic5 -x ./ui/ecc.ui -o ./ui/ecc.py
```

- Thêm hai dòng sau vào đầu file "ecc.py":

```

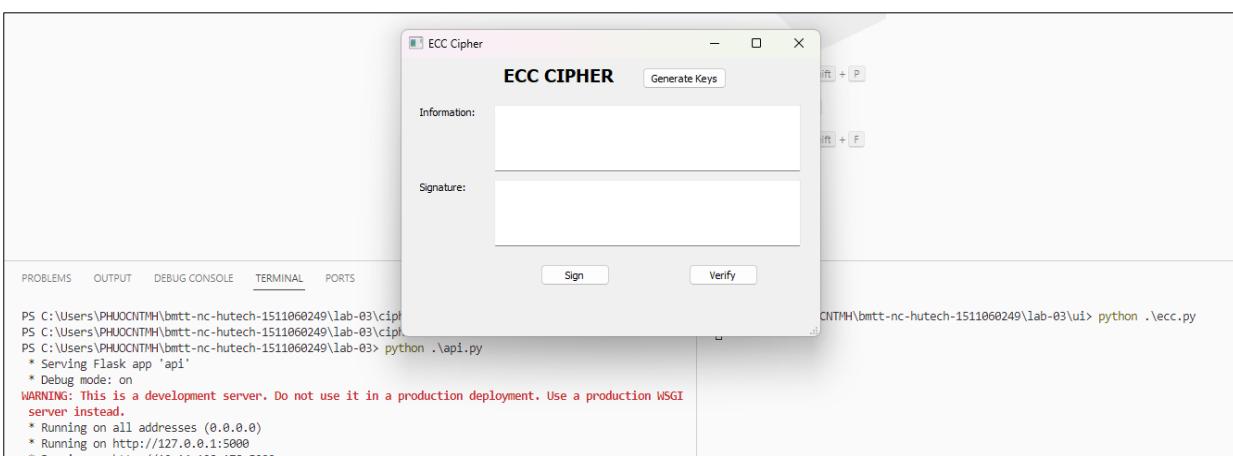
ecc.py  X

lab-03 > ui > ecc.py > Ui_MainWindow > setupUi
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file './ui/ecc.ui'
4  #
5  # Created by: PyQt5 UI code generator 5.15.10
6  #
7  # WARNING: Any manual changes made to this file will be lost when pyuic5 is
8  # run again. Do not edit this file unless you know what you are doing.
9
10
11 from PyQt5 import QtCore, QtGui, QtWidgets
12 import os
13 os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = "../platforms"


```

- Kiểm tra thực thi của giao diện:

```
python .\ecc.py
```



- Vừa rồi, chúng ta đã làm xong front-end, tiếp theo, chúng ta sẽ làm phần back-end.
- Trong folder "lab-03" tạo file "ecc_cipher.py" (Lưu ý: Kiểm tra tên của các button cho đúng).

```

ecc_cipher.py  X

lab-03 > ecc_cipher.py > ...
1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
3 from ui.ecc import Ui_MainWindow
4 import requests
5


```

```

6   class MyApp(QMainWindow):
7       def __init__(self):
8           super().__init__()
9           self.ui = Ui_MainWindow()
10          self.ui.setupUi(self)
11          self.ui.btn_gen_keys.clicked.connect(self.call_api_gen_keys)
12          self.ui.btn_sign.clicked.connect(self.call_api_sign)
13          self.ui.btn_verify.clicked.connect(self.call_api_verify)
14
15      def call_api_gen_keys(self):
16          url = "http://127.0.0.1:5000/api/ecc/generate_keys"
17          try:
18              response = requests.get(url)
19              if response.status_code == 200:
20                  data = response.json()
21                  msg = QMessageBox()
22                  msg.setIcon(QMessageBox.Information)
23                  msg.setText(data["message"])
24                  msg.exec_()
25              else:
26                  print("Error while calling API")
27          except requests.exceptions.RequestException as e:
28              print("Error: %s" % e.message)
29

```

```

30      def call_api_sign(self):
31          url = "http://127.0.0.1:5000/api/ecc/sign"
32          payload = {
33              "message": self.ui.txt_info.toPlainText(),
34          }
35          try:
36              response = requests.post(url, json=payload)
37              if response.status_code == 200:
38                  data = response.json()
39                  self.ui.txt_sign.setText(data["signature"])
40
41                  msg = QMessageBox()
42                  msg.setIcon(QMessageBox.Information)
43                  msg.setText("Signed Successfully")
44                  msg.exec_()
45              else:
46                  print("Error while calling API")
47          except requests.exceptions.RequestException as e:
48              print("Error: %s" % e.message)
49

```

```

50     def call_api_verify(self):
51         url = "http://127.0.0.1:5000/api/ecc/verify"
52         payload = {
53             "message": self.ui.txt_info.toPlainText(),
54             "signature": self.ui.txt_sign.toPlainText()
55         }
56         try:
57             response = requests.post(url, json=payload)
58             if response.status_code == 200:
59                 data = response.json()
60                 if (data["is_verified"]):
61                     msg = QMessageBox()
62                     msg.setIcon(QMessageBox.Information)
63                     msg.setText("Verified Successfully")
64                     msg.exec_()
65                 else:
66                     msg = QMessageBox()
67                     msg.setIcon(QMessageBox.Information)
68                     msg.setText("Verified Fail")
69                     msg.exec_()
70             else:
71                 print("Error while calling API")
72         except requests.exceptions.RequestException as e:
73             print("Error: %s" % e.message)
74
75     if __name__ == "__main__":
76         app = QApplication(sys.argv)
77         window = MyApp()
78         window.show()
79         sys.exit(app.exec_())

```

- Tiến hành kiểm tra ứng dụng:
 - Bước 1: Split Terminal của VS Code, di chuyển vào folder “lab-03” và chạy file “api.py”.

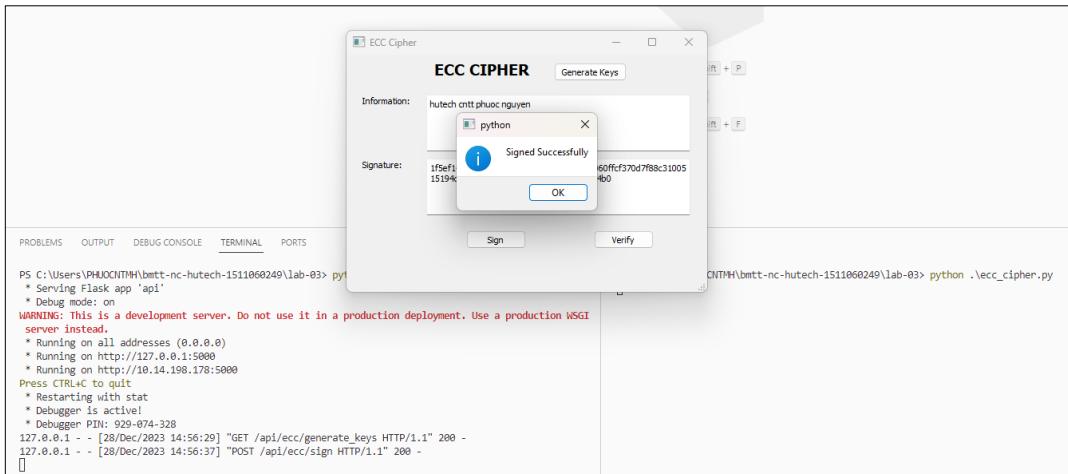
```
python .\api.py
```

- Bước 2: Tại Terminal của “lab-03”. Tiến hành chạy file “ecc_cipher.py”.

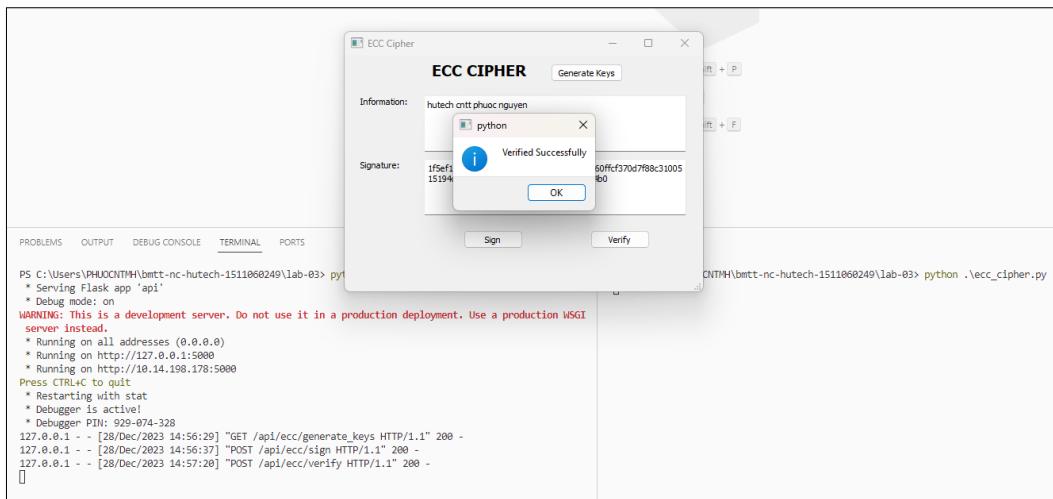
```
python .\ecc_cipher.py
```

- Bước 3: Giao diện ứng dụng xuất hiện, tiến hành kiểm tra các chức năng tạo khoá, ký, xác thực chữ ký.

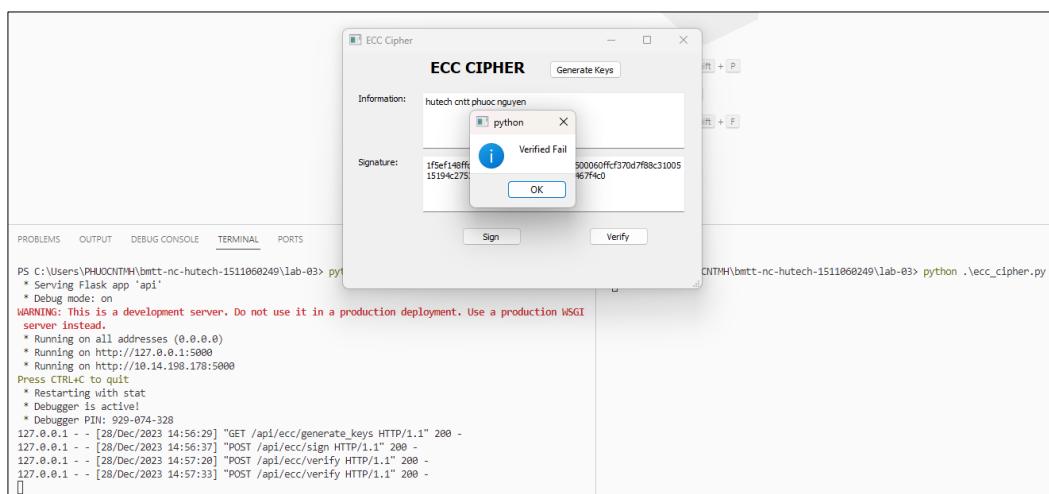
- Kết quả kiểm tra tạo chữ ký:



- Kết quả kiểm tra xác minh chữ ký:



- Thay đổi chữ ký, kiểm tra lại:



128

BÀI 3: LẬP TRÌNH GIAO DIỆN ỨNG DỤNG BẢO MẬT

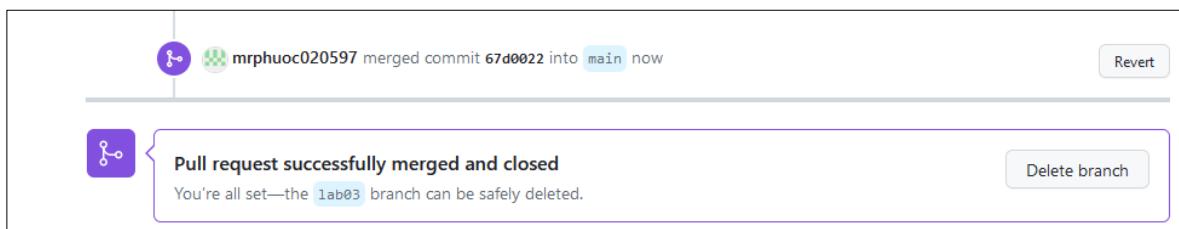
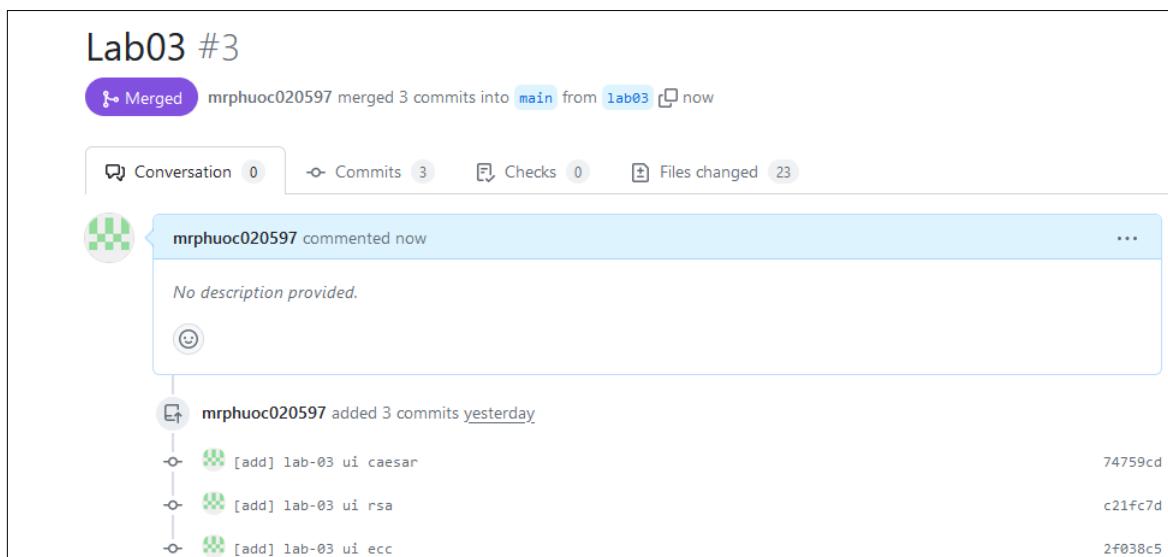
- Commit code:

```
git add .
git commit -m "[add] lab-03 ui ecc"
```

- Push các thay đổi lên remote repo:

```
git push origin lab03
```

- Tiến hành tạo PR từ nhánh lab03 về nhánh main. Review, kiểm tra đủ số lượng file, sau đó tiến hành “Merge pull request”.



3.6 BÀI TẬP MỞ RỘNG

- **Câu 01:** Thực hiện tạo giao diện trên desktop demo cho các mật mã: Vigenère, Rail Fence, Playfair, Transposition.
- **Câu 02:** Thực hiện tạo giao diện tạo chữ ký số và xác minh chữ ký của hai mật mã RSA và ECC trên cùng một màn hình.