Laporan Tugas Kecil 4

IF4020 Kriptografi

Semester 2021/2022

Dibuat oleh :

Kinantan Arya Bagaspati - 13519044

Christopher Chandrasaputra - 13519074

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bab 1. Deskripsi Masalah

Tugas Kecil 4 IF4020 Kriptografi semester 1 tahun 2021/2022 adalah membuat program yang dapat melakukan enkripsi dan dekripsi kunci publik dengan spesifikasi sebagai berikut:

1. Program terdiri dari:
    a. pembangkitan kunci privat dan kunci publik
    Kunci publik dan kunci privat dapat disimpan dalam file terpisah (misalnya *.pub dan *.pri)
    b. b. Enkripsi/dekripsi file
    Masukan: pesan, kunci privat/publik (browing atau diketik nilai kuncinya)
2. Program memiliki editor tempat pengguna mengetikkan pesan atau meng-copy paste teks ke editor tersebut.
3. Program dapat mengenkripsi plainteks dengan RSA, ElGamal, Paillier, ECC
4. Program dapat mendekripsi cipherteks dengan RSA, ElGamal, Paillier, ECC
5. Program menampilkan cipherteks di layar.
6. Tipe integer yang digunakan adalah long integer (pilih salah satu):
    a. Tipe Long Integer yang disediakan pada setiap bahasa/kakas
    b. Tipe BigNum yang pustakanya dapat diunduh dari internet (atau disediakan kakas)
    c. Tipe LongLongInteger bentukan sendiri
7. Kode program dibuat sendiri (tidak boleh copy/paste dari internet, kecuali pustaka BigNum).

Bab 2. *Source Code*

2.1. Halaman Utama

```
def main():
    TOOLS = ['RSA','ElGamal','Paillier','ECC']
    LAYOUT = [
        [sg.Text('Pick Tools
:'),sg.OptionMenu(values=TOOLS,default_value='RSA',key='option'),sg.Button('Pick')]
    ]

    window = sg.Window('Asymmetric Cryptography Tools', layout=LAYOUT, size=(250,50))

    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Cancel':
            break
        if event=='Pick':
            if values['option'] == 'RSA':
                RSA_GUI()
            elif values['option'] == 'ElGamal':
                ElGamal_GUI()
            elif values['option'] == 'Paillier':
                Paillier_GUI()
            elif values['option'] == 'ECC':
                ECC_GUI()
    window.close()

if __name__ == '__main__':
    main()
```

2.2. RSA

2.2.1. *GUI*

```
def RSA_GUI():
    layout = [
        [sg.Text('Text :'),sg.InputText(key='text')],
        [sg.Text('n = '),sg.InputText(key='n',default_text='1')],
        [sg.Text('e = '),sg.InputText(key='e',default_text='1')],
        [sg.Text('d = '),sg.InputText(key='d',default_text='1')],
        [sg.Button('Generate Key'),sg.Checkbox('Save to file',key='save')],
        [sg.Button('Import Public
Key'),sg.InputText(default_text='./key/rsa_key.pub',disabled=True,key='imp_pub'),sg.FileBrow
se(initial_folder='./key/', file_types=(("Public Key Files","*.pub"),))],
        [sg.Button('Import Private
Key'),sg.InputText(default_text='./key/rsa_key.pri',disabled=True,key='imp_pri'),sg.FileBrow
se(initial_folder='./key/', file_types=(("Private Key Files","*.pri"),))],
        [sg.Button('Encrypt'),sg.Button('Decrypt')],
        [sg.Multiline(size=(100,60),disabled=True,key='res')]
```

```
    ]
    window = sg.Window(title=('RSA'), layout=layout, size=(700,800), modal=True)
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Cancel':
            break
        res = ''
        pub_key = {'n':int(values['n']),'e':int(values['e'])}
        pri_key = {'n':int(values['n']),'d':int(values['d'])}
        tools = rsa.RSA(pub_key,pri_key)
        if event == 'Generate Key':
            if values['save']:
                tn, te, td = tools.generate_pair(True)
            else:
                tn, te, td = tools.generate_pair(False)
            window.Element(key='n').Update(tn)
            window.Element(key='e').Update(te)
            window.Element(key='d').Update(td)
        elif event == 'Import Public Key':
            key = tools.open_key(values['imp_pub'])
            window.Element(key='n').Update(key['n'])
            window.Element(key='e').Update(key['e'])
        elif event == 'Import Private Key':
            key = tools.open_key(values['imp_pri'])
            window.Element(key='n').Update(key['n'])
            window.Element(key='d').Update(key['d'])
        elif event == 'Encrypt' and len(values['text'])>0:
            transformed_text = bytearray(values['text'].encode())
            ciphertext = tools.encrypt(transformed_text)
            res = base64.b64encode(ciphertext).decode('utf-8')
        elif event == 'Decrypt' and len(values['text'])>0:
            try:
                transformed_text = bytearray(base64.b64decode(values['text']))
                plaintext = tools.decrypt(transformed_text)
                res = plaintext.decode('utf-8')
            except:
                res = 'Wrong ciphertext code'
        window.Element(key='res').Update(res)
```

2.2.2. Algoritma

```
class RSA:
    '''
    RSA Class
    '''
    # CONSTANTS
    RSA_BIT_SIZE = 1024 # key n's size in bit
    RSA_BLOCK_SIZE = RSA_BIT_SIZE//8 # How many byte can be grouped = RSA_N_bit/8
    RSA_PAD_INFO = 4 # Allocate 4 byte for padding info (Max padding = 4-byte-value of byte)
```

```
    # KEYS
    public_key = {'n':1,'e':1}
    private_key = {'n':1,'e':1}

    def __init__(self, pub_key={'n':None,'e':None}, priv_key={'n':None,'e':None}):
        self.public_key = pub_key
        self.private_key = priv_key

    def encrypt(self, plaintext: bytearray):
        '''
        RSA Algorithm to encrypt plaintext
        input:
         plaintext (bytearray)
        output:
         ciphertext (bytearray)
        '''
        # Splitting and Adding Guard
        plain_blocks = [b'\x00' + plaintext[i:i+self.RSA_BLOCK_SIZE-1] for i in
range(0,len(plaintext),self.RSA_BLOCK_SIZE-1)]

        # Padding length check
        pad_length = self.RSA_BLOCK_SIZE-len(plain_blocks[-1])
        if pad_length:
            plain_blocks[-1] = b'\x00' * pad_length + plain_blocks[-1]

        # Converting blocks to integer
        plain_blocks = [int.from_bytes(byte, byteorder='big', signed=False) for byte in
plain_blocks]

        # Encrypting
        cipher_blocks = []
        for i in range(len(plain_blocks)):

cipher_blocks.append(pow(plain_blocks[i],self.public_key['e'],self.public_key['n']))

        # Converting blocks to Byte
        cipher_blocks = [block.to_bytes(length=self.RSA_BLOCK_SIZE,
byteorder='big',signed=False) for block in cipher_blocks]

        # Generating ciphertext
        ciphertext = b''
        for block in cipher_blocks:
            ciphertext += block
        ciphertext +=
pad_length.to_bytes(length=self.RSA_PAD_INFO,byteorder='big',signed=False)

        return bytearray(ciphertext)

    def decrypt(self,ciphertext: bytearray):
        '''
        RSA Algorithm to decrypt ciphertext
```

```
        input:
         ciphertext (bytearray)
        output:
         plaintext (bytearray)
        '''
        # Splitting ciphertext with padding info
        cipher_blocks, padding = ciphertext[:-self.RSA_PAD_INFO],
int.from_bytes(ciphertext[-self.RSA_PAD_INFO:],byteorder='big',signed=False)

        # Splitting blocks
        cipher_blocks = [cipher_blocks[i:i+self.RSA_BLOCK_SIZE] for i in
range(0,len(cipher_blocks),self.RSA_BLOCK_SIZE)]

        # Converting blocks to integer
        cipher_blocks = [int.from_bytes(byte, byteorder='big', signed=False) for byte in
cipher_blocks]


        # Decrypting
        plain_blocks = []
        for i in range(len(cipher_blocks)):

plain_blocks.append(pow(cipher_blocks[i],self.private_key['d'],self.private_key['n']))

        # Converting blocks to Byte
        plain_blocks = [block.to_bytes(length=self.RSA_BLOCK_SIZE,
byteorder='big',signed=False) for block in plain_blocks]

        # Removing padding
        plain_blocks[-1] = plain_blocks[-1][padding:]

        # Removing guard
        plain_blocks = [block[1:] for block in plain_blocks]

        # Generating plaintext
        plaintext = b''
        for block in plain_blocks:
            plaintext += block

        return bytearray(plaintext)

    def generate_pair(self,save=False):
        '''
        Generate RSA key pair with save option
        if safe is True, write RSA key pair to
        folder ./key/
        input:
         save (boolean)
        output:
         n (int)
         e (int)
```

```
         d (int)
        '''
        p = gen(self.RSA_BIT_SIZE//2)
        q = gen(self.RSA_BIT_SIZE//2)
        n = p*q
        tot_n = (p-1)*(q-1)

        # GENERATING PUBLIC KEY
        def coprime(a,b):
            return gcd(a,b) == 1

        e = randrange(2,floor(sqrt(tot_n)))
        while not coprime(e,tot_n):
            e += 1

        # GENERATING PRIVATE KEY
        d = pow(e,-1,tot_n)

        # Creating Key Dictionary
        pub_key = {'n':n,'e':e}
        pri_key = {'n':n,'d':d}

        if save:
            json.dump(pub_key, open('./key/rsa_key.pub','w'))
            json.dump(pri_key, open('./key/rsa_key.pri','w'))

        self.public_key = pub_key
        self.private_key = pri_key

        return n, e, d

    def open_key(self, dir:str):
        '''
        Importing RSA key from directory
        input:
         dir (string)
        output:
         dictionary of {n,e} or dictionary of {n,d}
        '''
        with open(dir) as f:
            data = f.read()
            key = json.loads(data)
            if dir.endswith('.pub'):
                self.public_key = key
            elif dir.endswith('.pri'):
                self.private_key = key
            return key

    def set_n(self,n):
        '''
        Setting n value of public/private key
```

```
        input:
         n (int)
        '''
        self.private_key['n'] = n
        self.public_key['n'] = n

    def set_e(self,e):
        '''
        Setting n value of public key
        input:
         e (int)
        '''
        self.public_key['e'] = e

    def set_d(self,d):
        '''
        Setting n value of private key
        input:
         d (int)
        '''
        self.private_key['d'] = d
```

2.3. ElGamal

2.3.1. *GUI*

```
def ElGamal_GUI():
    layout = [
        [sg.Text('Text :'),sg.InputText(key='text')],
        [sg.Text('g = '),sg.InputText(key='g',default_text='2')],
        [sg.Text('x = '),sg.InputText(key='x',default_text='2')],
        [sg.Text('p = '),sg.InputText(key='p',default_text='257')],
        [sg.Button('Generate Key'),sg.Checkbox('Save to file',key='save')],
        [sg.Button('Import Public
Key'),sg.InputText(default_text='./key/eg_key.pub',disabled=True,key='imp_pub'),sg.FileBrows
e(initial_folder='./key/', file_types=(("Public Key Files","*.pub"),))],
        [sg.Button('Import Private
Key'),sg.InputText(default_text='./key/eg_key.pri',disabled=True,key='imp_pri'),sg.FileBrows
e(initial_folder='./key/', file_types=(("Private Key Files","*.pri"),))],
        [sg.Button('Encrypt'),sg.Button('Decrypt')],
        [sg.Multiline(size=(100,60),disabled=True,key='res')]
    ]
    window = sg.Window(title=('Elgamal'), layout=layout, size=(700,800), modal=True)
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Cancel':
            break
        res = ''
        tools = eg.Elgamal(int(values['g']), int(values['x']), int(values['p']))
        if event == 'Generate Key':
```

```
            if values['save']:
                p, g, x = tools.generate_pair(True)
            else:
                p, g, x = tools.generate_pair(False)
            window.Element(key='p').Update(p)
            window.Element(key='g').Update(g)
            window.Element(key='x').Update(x)
        elif event == 'Import Public Key':
            key = tools.open_key(values['imp_pub'])
            window.Element(key='p').Update(key['p'])
            window.Element(key='g').Update(key['g'])
        elif event == 'Import Private Key':
            key = tools.open_key(values['imp_pri'])
            window.Element(key='p').Update(key['p'])
            window.Element(key='x').Update(key['x'])
        elif event == 'Encrypt' and len(values['text'])>0:
            res = tools.encrypt_text(values['text'])
        elif event == 'Decrypt' and len(values['text'])>0:
            try:
                res = tools.decrypt_text(values['text'])
            except:
                res = 'Wrong ciphertext code'
        window.Element(key='res').Update(res)
    pass
```

2.3.2. Algoritma

```
class Elgamal:
    # Konstruktor
    def __init__(self, g, x, p = 257):
        '''
        Konstruktor ini mengasumsikan p prima, serta membuat field-field sesuai parameter
dan elgamal pada umumnya
        self.nByte adalah banyak Byte dari representasi biner p
        '''
        self.p = p
        self.g = g % p
        self.x = x % p
        self.y = fast_pow(g, x, p)
        self.nByte = 0
        p -= 1
        while (p>1):
            self.nByte += 1
            p //= 256
        print(self.nByte)
        self.EG_BIT_SIZE = 1024

    def encrypt_int(self, message_int, k) -> [int, int]:
        '''
```

```
        Enkripsi 1 karakter menjadi 2 karakter dengan parameter tambahan k, yang
diimplementasikan bil random
        '''
        return [fast_pow(self.g, k, self.p), message_int * fast_pow(self.y, k, self.p) %
self.p]

    def encrypt_text(self, message):
        '''
        Seperti ECC, terdapat opsi lain yakni message n karakter cukup dienkripsi jadi n+1
karakter saja dengan
        membangkitkan bil random k sekali saja.
        Namun implementasi ini tetap menggunakan 1 karakter -> 2 karakter
        '''
        chars = [message[self.nByte*i : min(self.nByte*(i+1), len(message))] for i in
range((len(message) + self.nByte-1)//self.nByte)]
        ints = [str_to_int(self.nByte, chars[i])+1 for i in range(len(chars))]
        result_int = [self.encrypt_int(ints[i], random.randrange(0, self.p-1)) for i in
range(len(ints))]
        print(result_int)
        result_char = [int_to_str(self.nByte, result_int[i//2][i%2] - 1) for i in
range(2*len(result_int))]
        return ''.join(result_char)

    def decrypt_int(self, a, b):
        '''
        Dekripsi sesuai metode elgamal
        '''
        return b * inv_mod(fast_pow(a, self.x, self.p), self.p) % self.p

    def decrypt_text(self, message):
        '''
        Dekripsi sesuai metode enkripsi, memanfaatkan str_to_int, int_to_str,
        dan memisah text jadi string-string sepanjang self.nByte
        '''
        chars = [message[self.nByte*i : min(self.nByte*(i+1), len(message))] for i in
range((len(message) + self.nByte-1)//self.nByte)]
        ints = [str_to_int(self.nByte, chars[i])+1 for i in range(len(chars))]
        result_int = [self.decrypt_int(ints[2*i], ints[2*i+1]) for i in
range(len(chars)//2)]
        print(result_int)
        result_char = [int_to_str(self.nByte, result_int[i]-1) for i in
range(len(result_int))]
        return ''.join(result_char)

    def generate_pair(self,save=False):
        '''
        Membangkitkan prima 1024 bit dengan fungsi dari primeGenerator.py
        '''
        p = gen(self.EG_BIT_SIZE)

        g = random.randrange(2, p-1)
```

```
        x = random.randrange(2, p-2)
        y = fast_pow(g, x, p)

        # Creating Key Dictionary
        pub_key = {'p':p,'g':g, 'y': y}
        pri_key = {'p':p,'x':x}

        if save:
            json.dump(pub_key, open('./key/eg_key.pub','w'))
            json.dump(pri_key, open('./key/eg_key.pri','w'))

        self.p = p
        self.g = g
        self.x = x
        self.y = y

        return p, g, x

    def open_key(self, dir:str):
        with open(dir) as f:
            data = f.read()
            key = json.loads(data)
            if dir.endswith('.pub'):
                self.p = key['p']
                self.g = key['g']
                self.y = key['y']
            elif dir.endswith('.pri'):
                self.p = key['p']
                self.x = key['x']
            return key
```

2.4. Paillier

2.4.1. *GUI*

```
def Paillier_GUI():
    layout = [
        [sg.Text('Text :'),sg.InputText(key='text')],
        [sg.Text('g = '),sg.InputText(key='g',default_text='1')],
        [sg.Text('n = '),sg.InputText(key='n',default_text='1')],
        [sg.Text('lambda = '),sg.InputText(key='lmd',default_text='1')],
        [sg.Text('mu = '),sg.InputText(key='mu',default_text='1')],
        [sg.Button('Generate Key'),sg.Checkbox('Save to file',key='save')],
        [sg.Button('Import Public
Key'),sg.InputText(default_text='./key/paillier_key.pub',disabled=True,key='imp_pub'),sg.Fil
eBrowse(initial_folder='./key/', file_types=(("Public Key Files","*.pub"),))],
        [sg.Button('Import Private
Key'),sg.InputText(default_text='./key/paillier_key.pri',disabled=True,key='imp_pri'),sg.Fil
eBrowse(initial_folder='./key/', file_types=(("Private Key Files","*.pri"),))],
        [sg.Button('Encrypt'),sg.Button('Decrypt')],
```

```
        [sg.Multiline(size=(100,60),disabled=True,key='res')]
    ]
    window = sg.Window(title=('Paillier'), layout=layout, size=(700,800), modal=True)
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Cancel':
            break
        res = ''
        pub_key = {'g':int(values['g']),'n':int(values['n'])}
        pri_key =
{'g':int(values['g']),'n':int(values['n']),'lmd':int(values['lmd']),'mu':int(values['mu'])}
        tools = pl.Paillier(pub_key,pri_key)
        if event == 'Generate Key':
            if values['save']:
                pub, pri = tools.generate_pair(True)
            else:
                pub, pri = tools.generate_pair(False)
            window.Element(key='g').Update(pub['g'])
            window.Element(key='n').Update(pub['n'])
            window.Element(key='lmd').Update(pri['lmd'])
            window.Element(key='mu').Update(pri['mu'])
        elif event == 'Import Public Key':
            key = tools.open_key(values['imp_pub'])
            window.Element(key='g').Update(key['g'])
            window.Element(key='n').Update(key['n'])
        elif event == 'Import Private Key':
            key = tools.open_key(values['imp_pri'])
            window.Element(key='g').Update(key['g'])
            window.Element(key='n').Update(key['n'])
            window.Element(key='lmd').Update(key['lmd'])
            window.Element(key='mu').Update(key['mu'])
        elif event == 'Encrypt' and len(values['text'])>0:
            transformed_text = bytearray(values['text'].encode())
            ciphertext = tools.encrypt(transformed_text)
            res = base64.b64encode(ciphertext).decode('utf-8')
        elif event == 'Decrypt' and len(values['text'])>0:
            try:
                transformed_text = bytearray(base64.b64decode(values['text']))
                plaintext = tools.decrypt(transformed_text)
                res = plaintext.decode('utf-8')
            except:
                res = 'Wrong ciphertext code'
        window.Element(key='res').Update(res)
```

2.4.2. Algoritma

```
class Paillier:
    PAIL_BIT_SIZE = 256
    PAIL_PAD_INFO = 4 # Reserve 4 byte for padding info
    public_key = {'g':1,'n':1}
```

```
    private_key = {'g':1,'n':1,'lmd':1,'mu':1}

    def __init__(self,pub_key={'g':1,'n':1},priv_key={'g':1,'n':1,'lmd':1,'mu':1}):
        self.public_key = pub_key
        self.private_key = priv_key

    def get_int_byte_length(self,num):
        return 1 if not num else int(log(num,256)) + 1

    def encrypt(self,plaintext:bytearray):
        '''
        Paillier Algorithm to encrypt plaintext
        input:
         plaintext (bytearray)
        output:
         ciphertext (bytearray)
        '''
        block_length = self.get_int_byte_length(self.public_key['n']**2)
        group_length = block_length//2

        # Splitting into groups
        plain_blocks = [plaintext[i:i+group_length] for i in
range(0,len(plaintext),group_length)]

        # Getting padding info
        pad_length = group_length-len(plain_blocks[-1])

        # Converting blocks to integer
        plain_blocks = [int.from_bytes(byte, byteorder='big', signed=False) for byte in
plain_blocks]

        # Encrypting
        cipher_blocks = []
        for block in plain_blocks:
            cipher_blocks.append(self.encrypt_block(block))

        # Converting blocks to Byte
        cipher_blocks = [block.to_bytes(length=block_length, byteorder='big',signed=False)
for block in cipher_blocks]

        # Generating ciphertext
        ciphertext = b''
        for block in cipher_blocks:
            ciphertext += block
        ciphertext +=
pad_length.to_bytes(length=self.PAIL_PAD_INFO,byteorder='big',signed=False)

        return ciphertext

    def decrypt(self,ciphertext:bytearray):
        '''
```

```
        Paillier Algorithm to decrypt plaintext
        input:
         ciphertext (bytearray)
        output:
         plaintext (bytearray)
        '''
        block_length = self.get_int_byte_length(self.private_key['n']**2)
        group_length = block_length//2
        # Splitting ciphertext with padding info
        cipher_blocks, padding = ciphertext[:-self.PAIL_PAD_INFO],
int.from_bytes(ciphertext[-self.PAIL_PAD_INFO:],byteorder='big',signed=False)

        # Splitting blocks
        cipher_blocks = [cipher_blocks[i:i+block_length] for i in
range(0,len(cipher_blocks),block_length)]

        # Converting blocks to integer
        cipher_blocks = [int.from_bytes(byte, byteorder='big', signed=False) for byte in
cipher_blocks]

        # Decrypting
        plain_blocks = []
        for block in cipher_blocks:
            plain_blocks.append(self.decrypt_block(block))

        # Converting blocks to Byte
        plain_blocks = [block.to_bytes(length=group_length, byteorder='big',signed=False)
for block in plain_blocks]

        # Removing padding
        plain_blocks[-1] = plain_blocks[-1][padding:]

        # Generating plaintext
        plaintext = b''
        for block in plain_blocks:
            plaintext += block

        return bytearray(plaintext)

    def encrypt_block(self,plain_block):
        '''
        Paillier Algorithm to encrypt block (in integer)
        input:
         plain_block (int)
        output:
         cipher_block (int)
        '''
        r = random.randrange(0,self.public_key['n']-1)
        while gcd(r,self.public_key['n']) != 1:
            r = random.randrange(0,self.public_key['n']-1)
        return (pow(self.public_key['g'],plain_block,self.public_key['n']**2) *
```

```
pow(r,self.public_key['n'],self.public_key['n']**2)) % self.public_key['n']**2

    def decrypt_block(self,cipher_block):
        '''
        Paillier Algorithm to decrypt block (in integer)
        input:
         cipher_block (int)
        output:
         plain_block (int)
        '''
        return (self.l(pow(cipher_block,self.private_key['lmd'], self.private_key['n']**2),
self.private_key['n'])*self.private_key['mu']) % self.private_key['n']

    def l(self,x,n):
        '''
        L(X) function for Paillier algorithm
        input:
         x (int)
         n (int)
        output:
         L(X) (int)
        '''
        return (x-1)//n

    def generate_pair(self,save=False):
        '''
        Generate Paillier key pair with save option
        if safe is True, write Paillier key pair to
        folder ./key/
        input:
         save (boolean)
        output:
         dictionary of {g,n}, dictionary of {g,n,lambda,mu}
        '''
        p,q = 0,0
        while True:
            p = gen(self.PAIL_BIT_SIZE)
            q = gen(self.PAIL_BIT_SIZE)
            if gcd(p*q,((p-1)*(q-1)))==1:
                break

        n = p*q
        lmd = lcm(p-1,q-1)
        g = random.randrange(2,(n**2)-1)

        x = pow(g,lmd,n**2)

        mu = pow(self.l(x,n),-1, n)

        pub_key = {"g":g, "n":n}
        pri_key = {"lmd":lmd, "mu":mu, "g":g, "n":n}
```

```
        if save:
            json.dump(pub_key, open('./key/paillier_key.pub','w'))
            json.dump(pri_key, open('./key/paillier_key.pri','w'))

        self.public_key = pub_key
        self.private_key = pri_key

        return pub_key,pri_key

    def open_key(self, dir:str):
        '''
        Importing Paillier key from directory
        input:
         save (boolean)
        output:
         dictionary of {g,n} or dictionary of {g,n,lambda,mu}
        '''
        with open(dir) as f:
            data = f.read()
            key = json.loads(data)
            if dir.endswith('.pub'):
                self.public_key = key
            elif dir.endswith('.pri'):
                self.private_key = key
            return key
```

2.5. ECC

2.5.1. *GUI*

```
def ECC_GUI():
    layout = [
        [sg.Text('Text :'),sg.InputText(key='text')],
        [sg.Text('y**2 = x**3 + '), sg.InputText(key='a',default_text='1'), sg.Text('x + '),
sg.InputText(key='b', default_text='1')],
        [sg.Text('Base Point = '),sg.InputText(key='g',default_text='0')],
        [sg.Text('x = '),sg.InputText(key='x',default_text='1')],
        [sg.Text('p = '),sg.InputText(key='p',default_text='32749')],
        [sg.Button('Generate Key'),sg.Checkbox('Save to file',key='save')],
        [sg.Button('Import Public
Key'),sg.InputText(default_text='./key/ecc_key.pub',disabled=True,key='imp_pub'),sg.FileBrow
se(initial_folder='./key/', file_types=(("Public Key Files","*.pub"),))],
        [sg.Button('Import Private
Key'),sg.InputText(default_text='./key/ecc_key.pri',disabled=True,key='imp_pri'),sg.FileBrow
se(initial_folder='./key/', file_types=(("Private Key Files","*.pri"),))],
        [sg.Button('Encrypt'),sg.Button('Decrypt')],
        [sg.Multiline(size=(100,60),disabled=True,key='res')]
    ]
    window = sg.Window(title=('ECC'), layout=layout, size=(700,800), modal=True)
```

```
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Cancel':
            break

        tools = ecc.ECC(int(values['a']),int(values['b']), values['g'], int(values['x']),
int(values['p']))
        res = tools.isViolate()

        if event == 'Generate Key':
            if values['save']:
                p, a, b, g, x = tools.generate_pair(True)
            else:
                p, a, b, g, x = tools.generate_pair(False)
            window.Element(key='p').Update(p)
            window.Element(key='a').Update(a)
            window.Element(key='b').Update(b)
            window.Element(key='g').Update(str(g[0]) + " " + str(g[1]))
            window.Element(key='x').Update(x)
        elif event == 'Import Public Key':
            key = tools.open_key(values['imp_pub'])
            window.Element(key='p').Update(key['p'])
            window.Element(key='a').Update(key['a'])
            window.Element(key='b').Update(key['b'])
            window.Element(key='g').Update(str(key['g'][0]) + " " + str(key['g'][1]))
        elif event == 'Import Private Key':
            key = tools.open_key(values['imp_pri'])
            window.Element(key='p').Update(key['p'])
            window.Element(key='x').Update(key['x'])
        elif res != '':
            res = res
        elif event == 'Encrypt' and len(values['text'])>0:
            res = tools.encrypt_text(values['text'])
        elif event == 'Decrypt' and len(values['text'])>0:
            res = tools.decrypt_text(values['text'])
            try:
                res = tools.decrypt_text(values['text'])
            except:
                res = 'Wrong ciphertext code'
        window.Element(key='res').Update(res)
```

2.5.2. Algoritma

```
class ECC:
    def __init__(self, a, b, g, x, p = 32749):
        '''
        Konstruktor kelas ini dibedakan untuk p < 32768 atau sebaliknya, sebut p kecil dan
besar
        Untuk p kecil, self.sqrt_mod[i] menyatakan y sehingga y**2 = i mod p
```

```
        Untuk p besar, hal diatas tidak bisa dihitung (terlalu lama), melainkan dihitung
banyak byte bilangan p
        Untuk p besar juga, simpan self.mul_base yang merupakan perkalian self.base dengan 0
hingga 255
        p sudah diasumsikan prima, g dapat berupa 2 bilangan sebagai point, atau 1 bilangan
sebagai generator
        '''
        self.p = p
        self.a = a % p
        self.b = b % p
        g = g.split(" ")

        if 2*p < 65536:
            self.sqrt_mod = [-1 for i in range(p)]
            for i in range(p):
                self.sqrt_mod[i*i%p] = i
        else:
            self.nByte = 0
            self.pow256 = 1
            while(p>0):
                self.nByte += 1
                p //= 256
                self.pow256 *= 256

        # print(g)
        if len(g) == 1:
            try:
                self.base = self.generate_point(int(g[0]))
            except:
                self.base = None
        elif len(g) == 2:
            try:
                self.base = [int(g[0]), int(g[1])]
            except:
                self.base = None
        else:
            self.base = None

        self.key_x = x
        if(self.base != None):
            self.key_y = self.mul_point(x, self.base)
            self.mul_base = [[-1,-1], self.base]
            for i in range(255):
                self.mul_base.append(self.mul_point(i+2, self.base))



    def isViolate(self):
        '''
        Menentukan apakah parameter kunci publik dan privat yang dimasukkan memenuhi batasan
atau tidak
```

```
        Dapat dilihat semua kasusnya dalam kode di bawah
        '''
        if (4*self.a*self.a*self.a + 27*self.b*self.b) % self.p == 0:
            return "4a**3 + 27b**2 == 0"
        if(self.base == None):
            return "Base point input not valid"
        xeq = self.base[0]
        yeq = self.base[1]
        if ((xeq*xeq*xeq + self.a*xeq + self.b - yeq*yeq) % self.p != 0):
            return "Base point is not in the curve"

        return ""

    def add_point(self, tuple1, tuple2):
        '''
        Menambahkan point berkoordinat tuple1 dan tuple2 sesuai definisi penambahan pada
kurva
        Point inf adalah [-1,-1]
        '''
        if (tuple1[0] < 0):
            return copy.deepcopy(tuple2)
        m = 0
        if(tuple1[0] == tuple2[0]):
            if(tuple1[1] == tuple2[1] and tuple1[1]>0):
                m_top = (3*tuple1[0]*tuple1[0] + self.a) % self.p
                m_bot = (2*tuple1[1]) % self.p
                m = m_top * inv_mod(m_bot, self.p) % self.p
            else:
                return [-1, -1]
        else:
            m_top = (tuple1[1] - tuple2[1] + self.p) % self.p
            m_bot = (tuple1[0] - tuple2[0] + self.p) % self.p
            m = m_top * inv_mod(m_bot, self.p) % self.p

        xr = (m*m-tuple1[0]-tuple2[0]+2*self.p) % self.p
        yr = ((tuple1[0]-xr+self.p) % self.p * m - tuple1[1] + self.p) % self.p
        return [xr, yr]

    def mul_point(self, scalar, tup):
        '''
        Mengalikan skalar scalar dengan point berkoordinat tup dengan penambahan biner pada
kurva
        Point inf adalah [-1,-1]
        '''
        tup_cpy = copy.deepcopy(tup)
        result = [-1, -1]
        while(scalar > 0):
            if(scalar%2):
                result = self.add_point(result, tup_cpy)
            scalar //= 2
            tup_cpy = self.add_point(tup_cpy, tup_cpy)
```

```
        return result

    def encrypt_int(self, message_int, k) -> [int, int]:
        '''
        Mengenkripsi 1 karakter menjadi 2 point sesuai prosedur ECC
        Bila p kecil membangkitkan point, sebaliknya ambil dari mul_base karena waktu
terbatas
        '''
        if(2*self.p < 65536):
            point = self.generate_point(message_int)
            return [self.mul_point(k, self.base), self.add_point(point, self.mul_point(k,
self.key_y))]
        else: # Big p, cannot generate
            point = self.mul_base[message_int+1]
            return [self.mul_point(k, self.base), self.add_point(point, self.mul_point(k,
self.key_y))]

    def encrypt_text(self, message):
        '''
        Tiap karakter dienkripsi jadi 2 point. Sebenarnya terdapat opsi lain diantaranya:
        - Tiap karakter jadi 1 point, dengan menambahkan k*base 1 point terdepan (tidak
perlu random tiap karakter, cukup sekali)
        - Fungsi mengembalikan points ketimbang characters
        '''
        chars = list(message)
        ints = [ord(chars[i]) for i in range(len(chars))]
        result_int = [self.encrypt_int(ints[i], random.randrange(0, self.p-1)) for i in
range(len(ints))]
        # print(result_int)
        result = ""
        if(2*self.p<65536):
            '''
            Tiap points dipetakan jadi 2 karakter saja cukup
            Ini dikarenakan 0 < x < 32768 dan hanya ada 2 kemungkinan y tiap nilai x
            Sehingga banyak kemungkinan kurang dari 65536 yakni tepat 2 byte atau 2 karakter
            Point [-1,-1] dipetakan ke 65535
            '''
            for i in range(len(result_int)):
                for j in range(2):
                    if(result_int[i][j][0] < 0):
                        result += chr(255)
                        result += chr(255)
                    else:
                        twochar = result_int[i][j][0]*2 + (result_int[i][j][1] >
(self.p//2))
                        result += chr(twochar // 256)
                        result += chr(twochar % 256)
        else:
            '''
            Tiap points dipetakan jadi nBytes karakter saja cukup, menggunakan fungsi
int_to_str
```

```
        '''
        for i in range(len(result_int)):
            # print(result_int[i])
            for j in range(2):
                for k in range(2):
                    if result_int[i][j][k]<0:
                        result += int_to_str(self.nByte, self.pow256-1)
                    else:
                        result += int_to_str(self.nByte, result_int[i][j][k])
    return result


def decrypt_int(self, int0, int1, int2, int3):
    '''
    Bagi kasus p kecil/besar, sesuaikan dengan enkripsi
    Bila p kecil, cukup kembalikan floor((x-1)/256)
    Bila p besar, kembalikan mul_base yang sesuai
    Untuk menegasikan tuple pertama cukup ganti ordinatnya(y) dengan p-y
    '''
    if 2*self.p < 65536:
        x1 = (int0*256 + int1)//2 % self.p
        y1 = self.sqrt_mod[(x1*x1*x1 + self.a*x1 + self.b) % self.p]
        if(int1%2):
            y1 = (self.p - y1) % self.p
        tuple1 = [x1, y1]
        if(int0*256 + int1 == 255*256 + 255):
            tuple1 = [-1, -1]

        x2 = (int2*256 + int3)//2 % self.p
        y2 = self.sqrt_mod[(x2*x2*x2 + self.a*x2 + self.b) % self.p]
        if(int3%2 == 0):
            y2 = (self.p - y2) % self.p
        tuple2 = [x2, y2]
        if(int2*256 + int3 == 255*256 + 255):
            tuple2 = [-1, -1]

        result = self.add_point(self.mul_point(self.key_x, tuple1), tuple2)
        return (result[0]-1) // (self.p//256)
    else:
        tuple1 = [(int0+1)%self.pow256-1, (int1+1)%self.pow256-1]
        tuple2 = [(int2+1)%self.pow256-1, (int3+1)%self.pow256-1]

        if(tuple1[1] >= 0):
            tuple1[1] = (self.p - tuple1[1]) % self.p

        result = self.add_point(self.mul_point(self.key_x, tuple1), tuple2)
        for i in range(256):
            if result == self.mul_base[i+1]:
                return i
        return -1
```

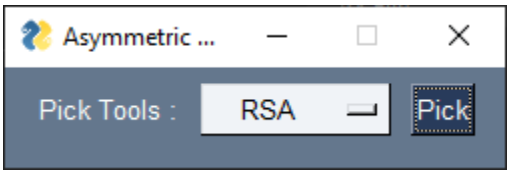
```
    def decrypt_text(self, message):
        '''
        Sesuaikan p kecil/besar dengan algoritma enkripsi
        Untuk p kecil, lakukan decrypt_int tiap 4 karakter, sebaliknya tiap 4*nByte karakter
dengan str_to_int
        '''
        if 2*self.p < 65536:
            chars = list(message)
            ints = [ord(chars[i]) for i in range(len(chars))]
            result_int = [self.decrypt_int(ints[4*i], ints[4*i+1], ints[4*i+2], ints[4*i+3])
for i in range(len(chars)//4)]
            # print(result_int)
            result_char = [chr(result_int[i]) for i in range(len(result_int))]
            return ''.join(result_char)
        else:
            chars = [message[self.nByte*i : min(self.nByte*(i+1), len(message))] for i in
range((len(message) + self.nByte-1)//self.nByte)]
            ints = [str_to_int(self.nByte, chars[i]) for i in range(len(chars))]

            result_int = [self.decrypt_int(ints[4*i], ints[4*i+1], ints[4*i+2], ints[4*i+3])
for i in range(len(chars)//4)]
            print(result_int)
            result = ""
            for i in range(len(result_int)):
                if result_int[i] < 0:
                    return "Wrong ciphertext"
                result += chr(result_int[i])
            return result

    def generate_point(self, m):
        '''
        Memilih point pada kurva dengan mencari solusi kurva dengan mencoba dari x=mk+1
hingga ketemu
        '''
        k = self.p // 256
        y = -1
        x = m*k
        while(y < 0):
            x += 1
            y = self.sqrt_mod[(x*x*x + self.a*x + self.b) % self.p]
        return [x, y]

    def generate_pair(self,save=False):
        '''
        Membangkitkan prima 1024 bit dengan fungsi dari primeGenerator.py
        '''
        p = gen(64)

        g = [random.randrange(0, p-1), random.randrange(0, p-1)]
        x = random.randrange(2, p-2)
        y = self.mul_point(x, g)
```

```python
        a = random.randrange(0, p-1)
        b = ((g[1]*g[1] - g[0]*g[0]*g[0] - a*g[0]) % p + p) % p

        # Creating Key Dictionary
        pub_key = {'p':p,'a': a, 'b': b, 'g':g, 'y': y}
        pri_key = {'p':p,'x':x}

        if save:
            json.dump(pub_key, open('./key/ecc_key.pub','w'))
            json.dump(pri_key, open('./key/ecc_key.pri','w'))

        self.p = p
        self.g = g
        self.a = a
        self.b = b
        self.x = x
        self.y = y

        return p, a, b, g, x

    def open_key(self, dir:str):
        with open(dir) as f:
            data = f.read()
            key = json.loads(data)
            if dir.endswith('.pub'):
                self.p = key['p']
                self.a = key['a']
                self.b = key['b']
                self.g = key['g']
            elif dir.endswith('.pri'):
                self.p = key['p']
                self.x = key['x']
            return key
```

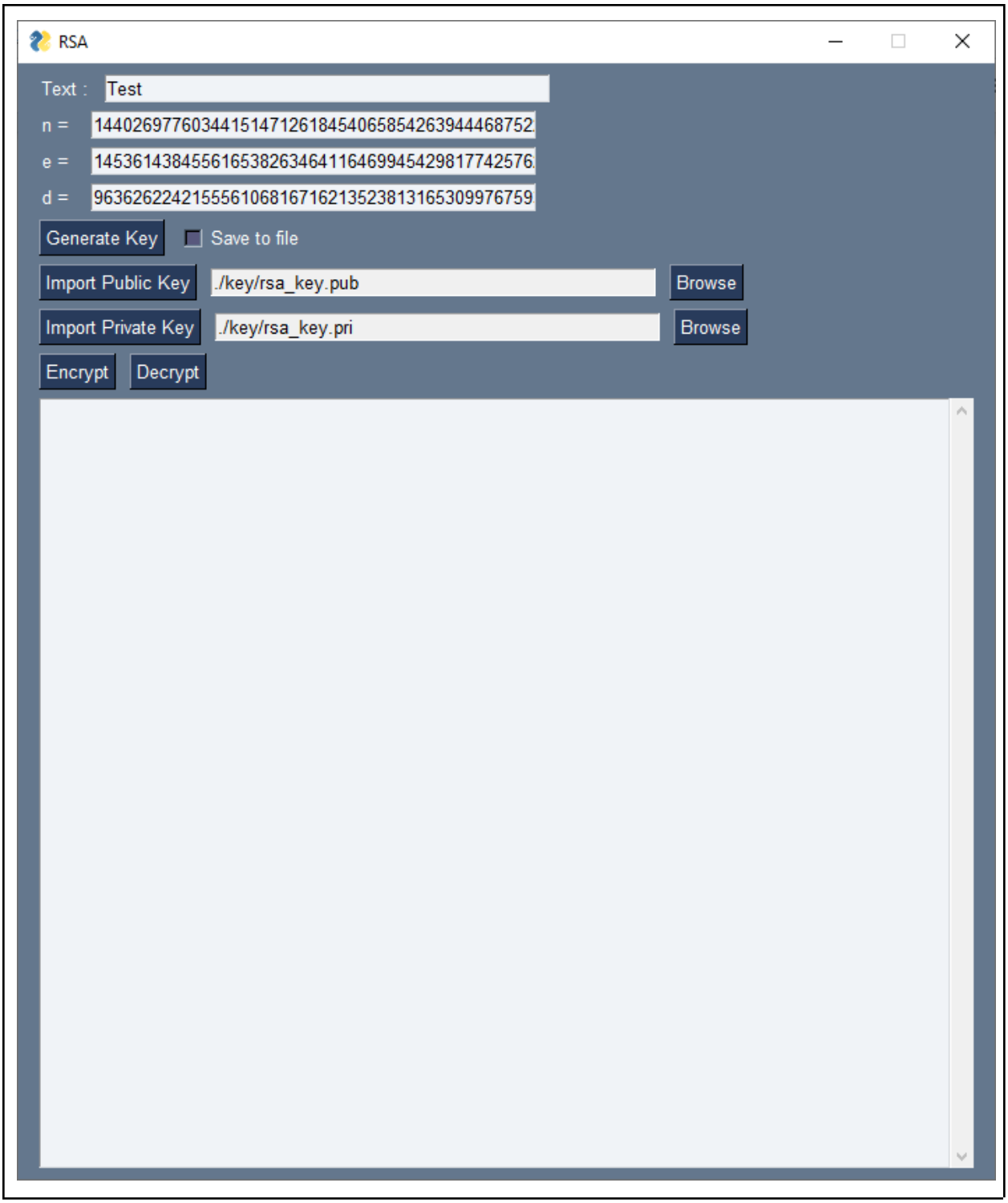
2.6. Utility

Bab 3. Tampilan Antarmuka

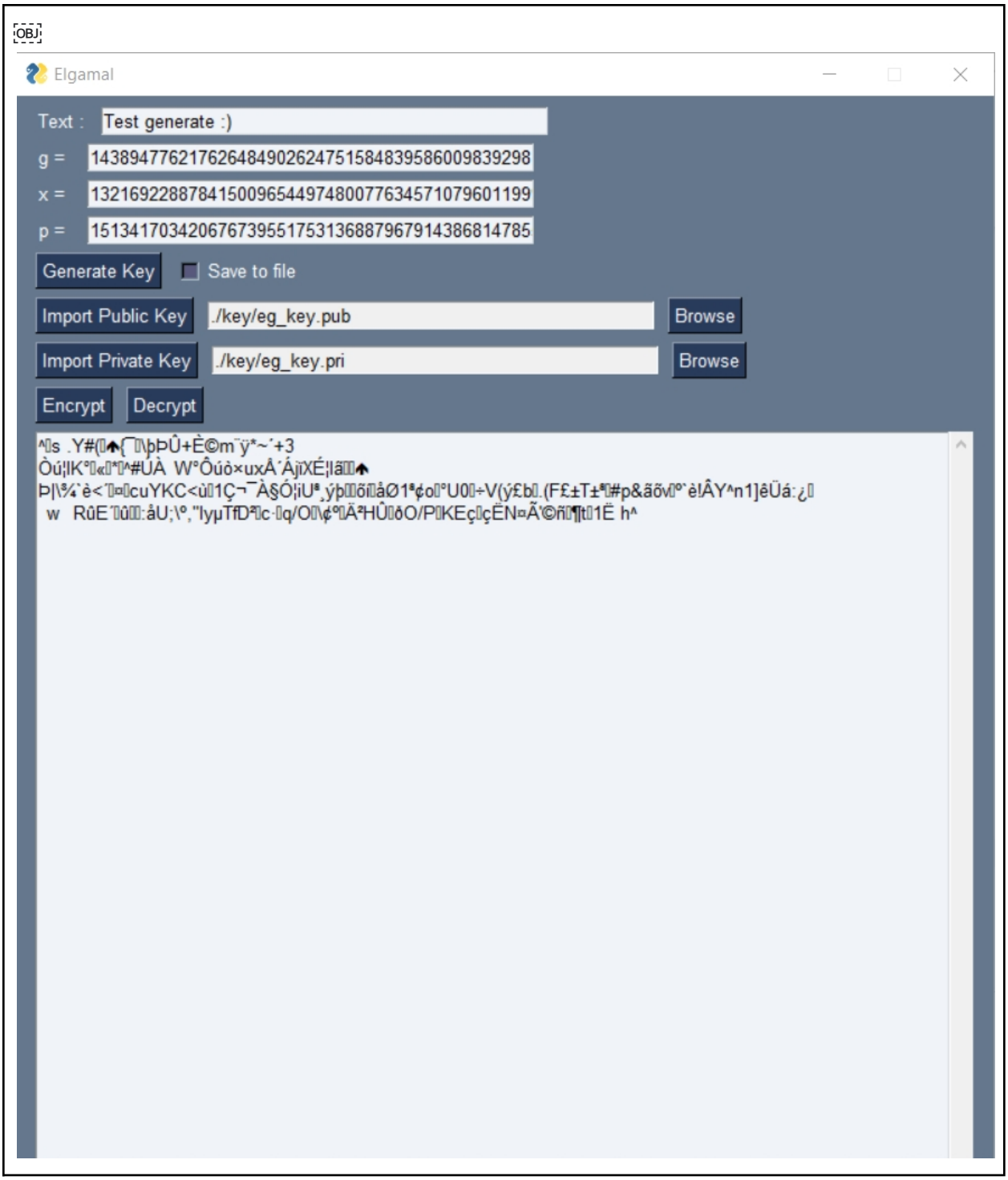
3.1. Halaman Utama



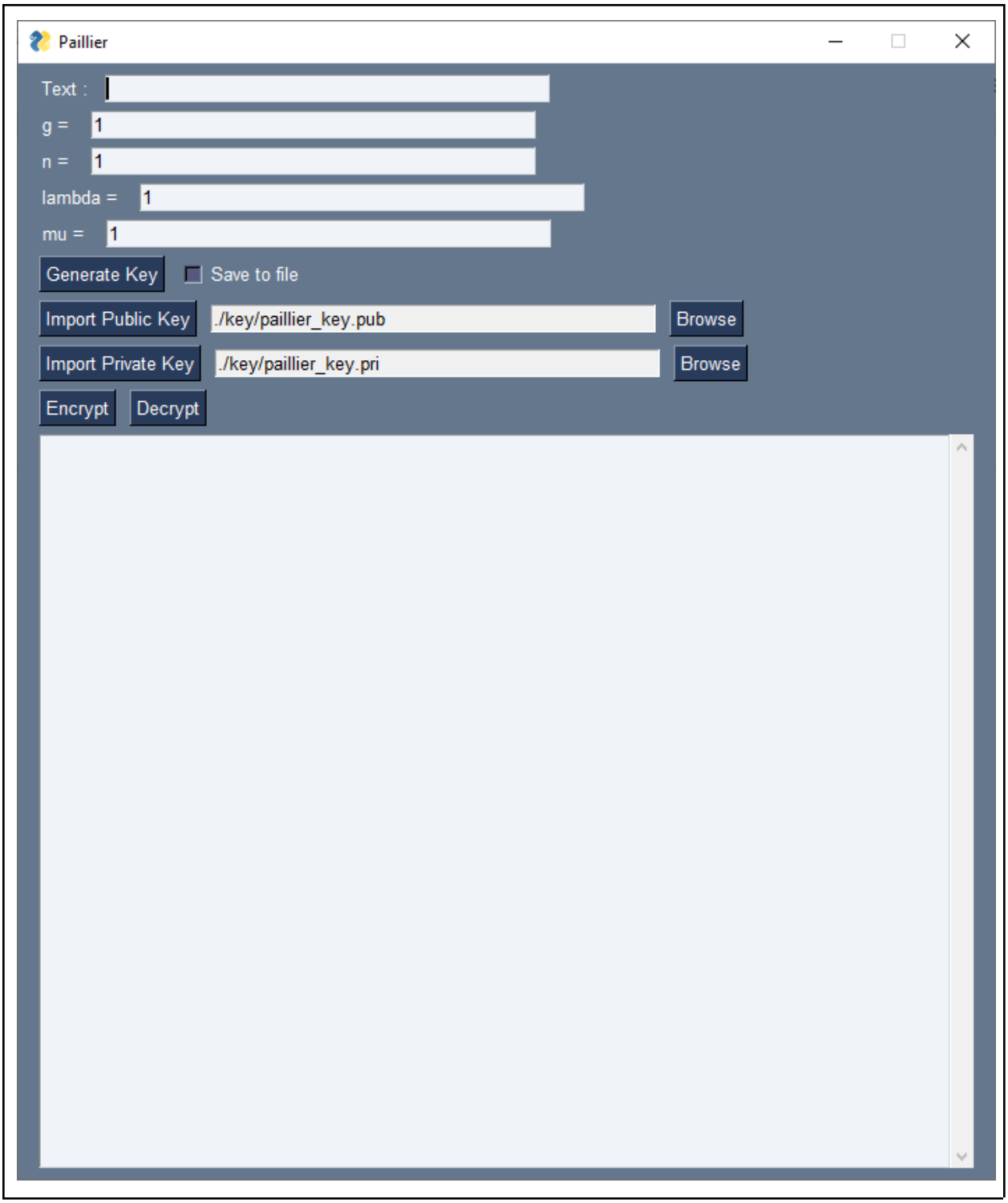
3.2. RSA


RSA
Text :
Test
n =
1440269776034415147126184540658542639444687 52
e =
1453614384556165382634641164699454298177425 76
d =
9636262242155561068167162135238131653099767 59
Generate Key
Save to file
Import Public Key
./key/rsa_key.pub
Browse
Import Private Key
./key/rsa_key.pri
Browse
Encrypt
Decrypt

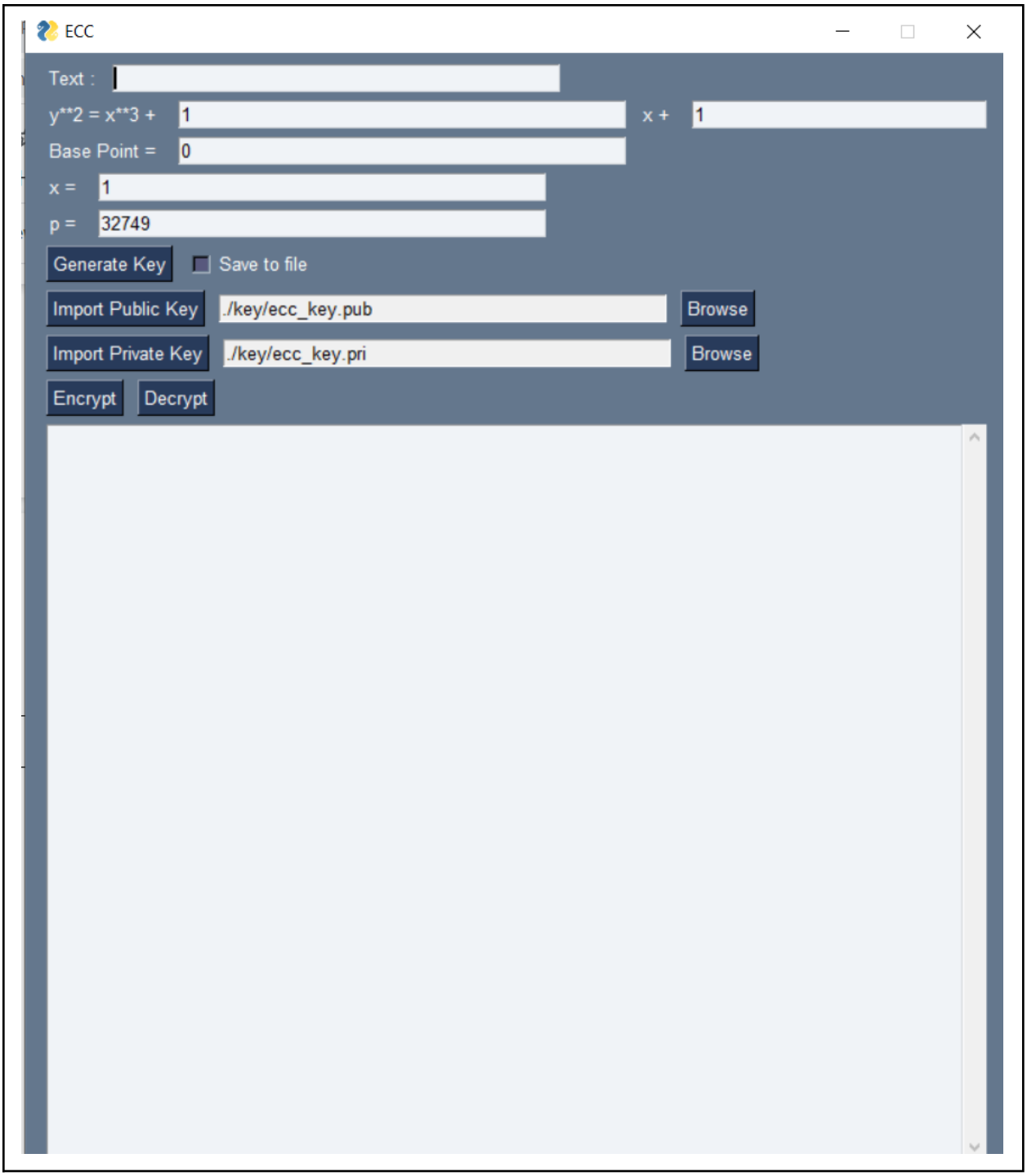
3.3. ElGamal


Elgamal
Text :
Test generate :)
g =
1438947762176264849026247515848395860098392 98
x =
1321692288784150096544974800776345710796011 99
p =
1513417034206767395517531368879679143868147 85
Generate Key
Save to file
Import Public Key
./key/eg_key.pub
Browse
Import Private Key
./key/eg_key.pri
Browse
Encrypt
Decrypt

3.4. Paillier


Paillier
Text :
g =
1
n =
1
lambda =
1
mu =
1
Generate Key
Save to file
Import Public Key
./key/paillier_key.pub
Browse
Import Private Key
./key/paillier_key.pri
Browse
Encrypt
Decrypt

3.5. ECC


ECC
Text :
y**2 = x**3 +
1
x +
1
Base Point =
0
x =
1
p =
32749
Generate Key
Save to file
Import Public Key
./key/ecc_key.pub
Browse
Import Private Key
./key/ecc_key.pri
Browse
Encrypt
Decrypt

Bab 4. Contoh

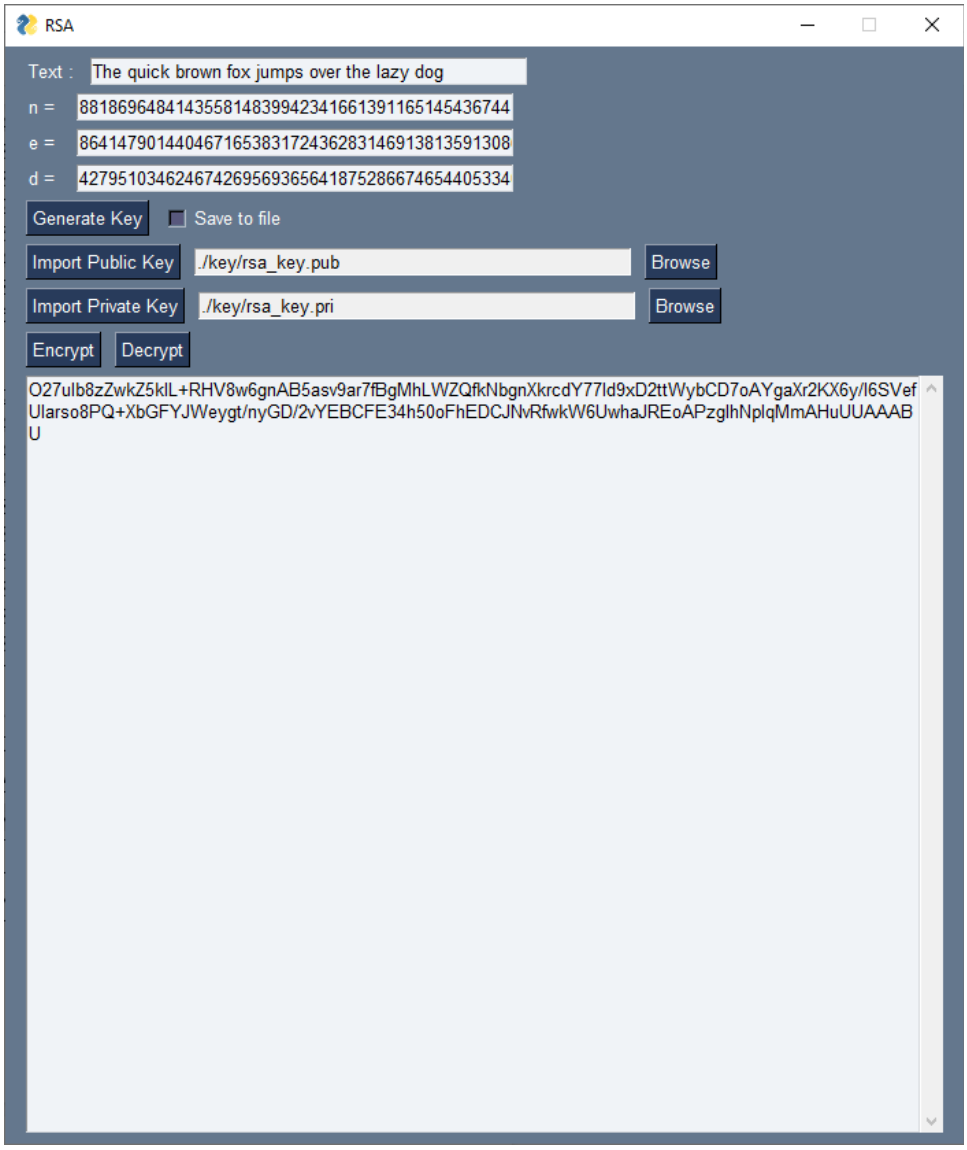
4.1. RSA

4.1.1. Uji Coba 1

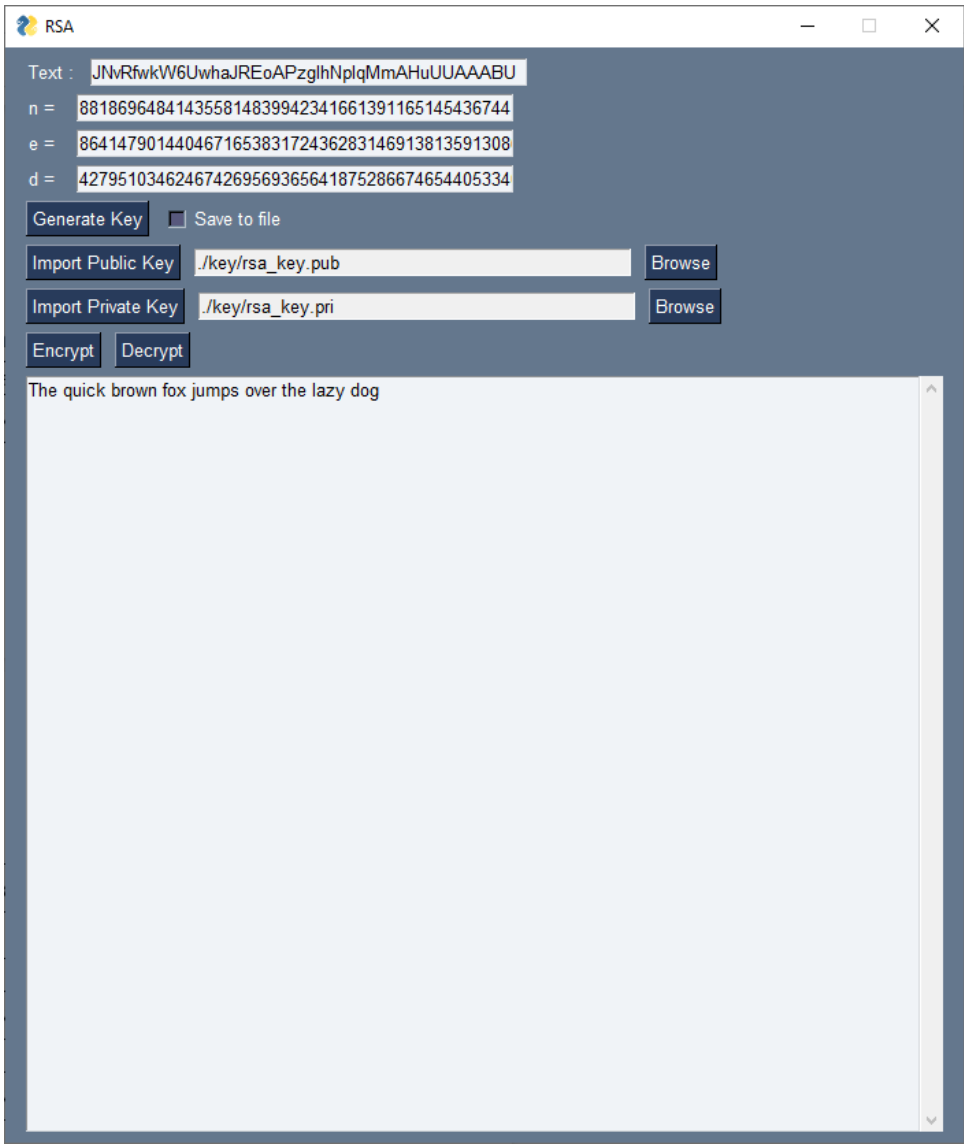
Plaintext : “The quick brown fox jumps over the lazy dog”
Generate Key :

Public Key	Private Key
n= 881869648414355814839942341661391165145436744162607510953112623974957443084628049712681777743006782200395331947199587154446130951415034242524181895519530061767467537042627945256232667337933661024641741709862428788670610359000840773315623420989940811600217337896003174542722666627511222550322169214706748784 23 e= 864147901440467165383172436283146913813591308679566976917901083244147704180600406851674182425940849853192121500357554969169332387193435667804168754103682 7	n= 881869648414355814839942341661391165145436744162607510953112623974957443084628049712681777743006782200395331947199587154446130951415034242524181895519530061767467537042627945256232667337933661024641741709862428788670610359000840773315623420989940811600217337896003174542722666627511222550322169214706748784 23 d= 427951034624674269569365641875286674654405334000964319474040938515176431619469928202268695186099115363842412781205493820331837676113874208896524305399761670032953028334614329922023535149711361830843181045454088073715270792100855063470726863922251604389879880470196167226990311111431663176535736762911280742 83

Ciphertext :
“O27uIb8zZwkZ5kIL+RHV8w6gnAB5asv9ar7fBgMhLWZQfkNbgnXkrcdY77ld9xD2ttWybCD7oAYgaXr2KX6y/I6SVefUlarso8PQ+XbGFYJWeygt/nyGD/2vYEBCFE34h50oFhEDCJNvRfwkW6UwhaJREoAPzglhNpIqMmAHuUUAAABU”



Plaintext : “The quick brown fox jumps over the lazy dog”



4.1.2. Uji Coba 2

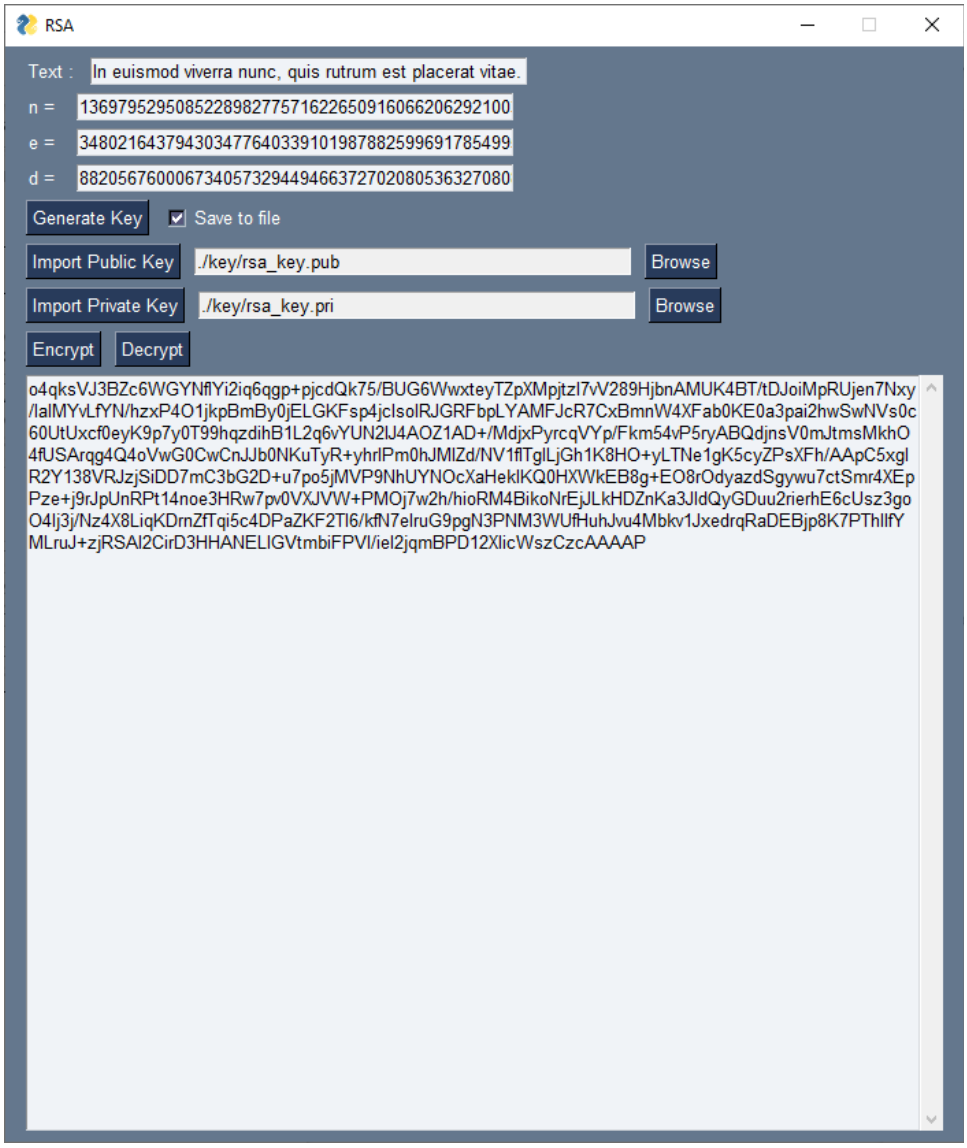
Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”

Input Key :

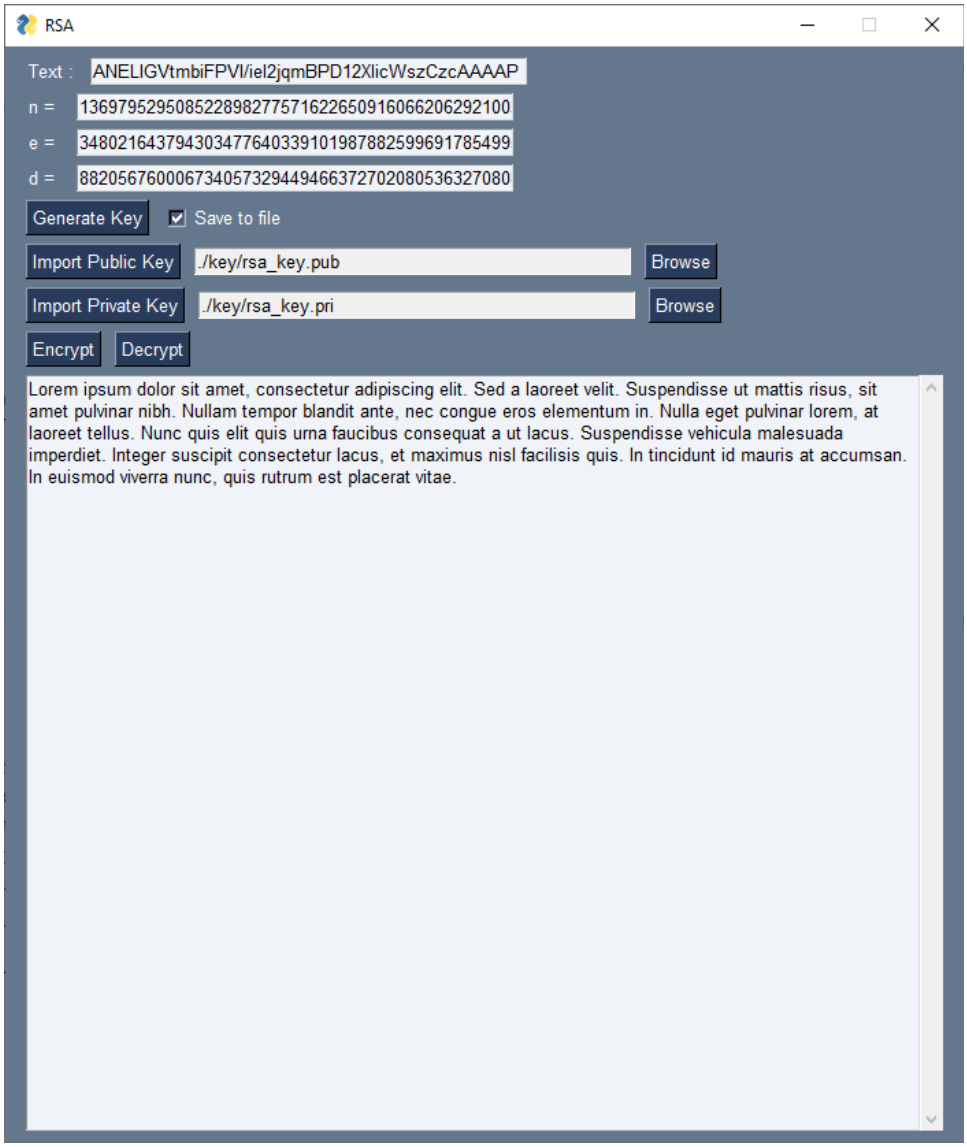
Public Key	Private Key
n= 13697952950852289827757162265091606620 62921003843838593081158957893982783509 46736952027793384422479187360976328152 02384995152419139611853301573626246418 58446324588901242800082870496751041512 24850459910616384792472638335473452147 51288685227185874528632225143403023182 93421803638055222040269985880974009618 24999 e= 34802164379430347764033910198788259969 17854995277980812360313752551047320517 74280063830828075484941461271836464125 84412026766563580382487691584945536709 57	n= 13697952950852289827757162265091606620 62921003843838593081158957893982783509 46736952027793384422479187360976328152 02384995152419139611853301573626246418 58446324588901242800082870496751041512 24850459910616384792472638335473452147 51288685227185874528632225143403023182 93421803638055222040269985880974009618 24999 d= 88205676000673405732944946637270208053 63270808074047308171599050427104259551 25557068415900530012815289167149445253 09039397834181370501751060140289215800 36121135890697621281276613955956602123 78652870671057951263953477330651213757 12021549066099979442206717622190109116 48859287077044959300832658674860961675 1253

Ciphertext :

“o4qksVJ3BZc6WGYNflYi2iq6qgp+pjcdQk75/BUG6WwxteyTZpXMpjtzI7vV289HjbnAMUK
4BT/tDJoiMpRUjen7Nxy/lalMYvLfYN/hzxP4O1jkpBmBy0jELGKFsp4jcIsolRJGRFbpLYAMF
JcR7CxBmnW4XFab0KE0a3pai2hwSwNVs0c60UtUxcf0eyK9p7y0T99hqzdihB1L2q6vYUN2I
J4AOZ1AD+/MdjxPyrcqVYp/Fkm54vP5ryABQdjnsV0mJtmsMkhO4fUSArqg4Q4oVwG0CwC
nJJb0NKuTyR+yhrlPm0hJMIZd/NV1flTgILjGh1K8HO+yLTNe1gK5cyZPsXFh/AApC5xglR2
Y138VRJzjSiDD7mC3bG2D+u7po5jMVP9NhUYNOcXaHekIKQ0HXWkEB8g+EO8rOdyazd
Sgywu7ctSmr4XEpPze+j9rJpUnRPt14noe3HRw7pv0VXJVW+PMOj7w2h/hioRM4BikoNrEjJ
LkHDZnKa3JIdQyGDuu2rierhE6cUsz3goO4Ij3j/Nz4X8LiqKDrnZfTqi5c4DPaZKF2TI6/kfN7e
IruG9pgN3PNM3WUfHuhJvu4Mbkv1JxedrqRaDEBjp8K7PThllfYMLruJ+zjRSAI2CirD3HHA
NELIGVtmbiFPVI/ieI2jqmBPD12XlicWszCzcAAAAP”



Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”



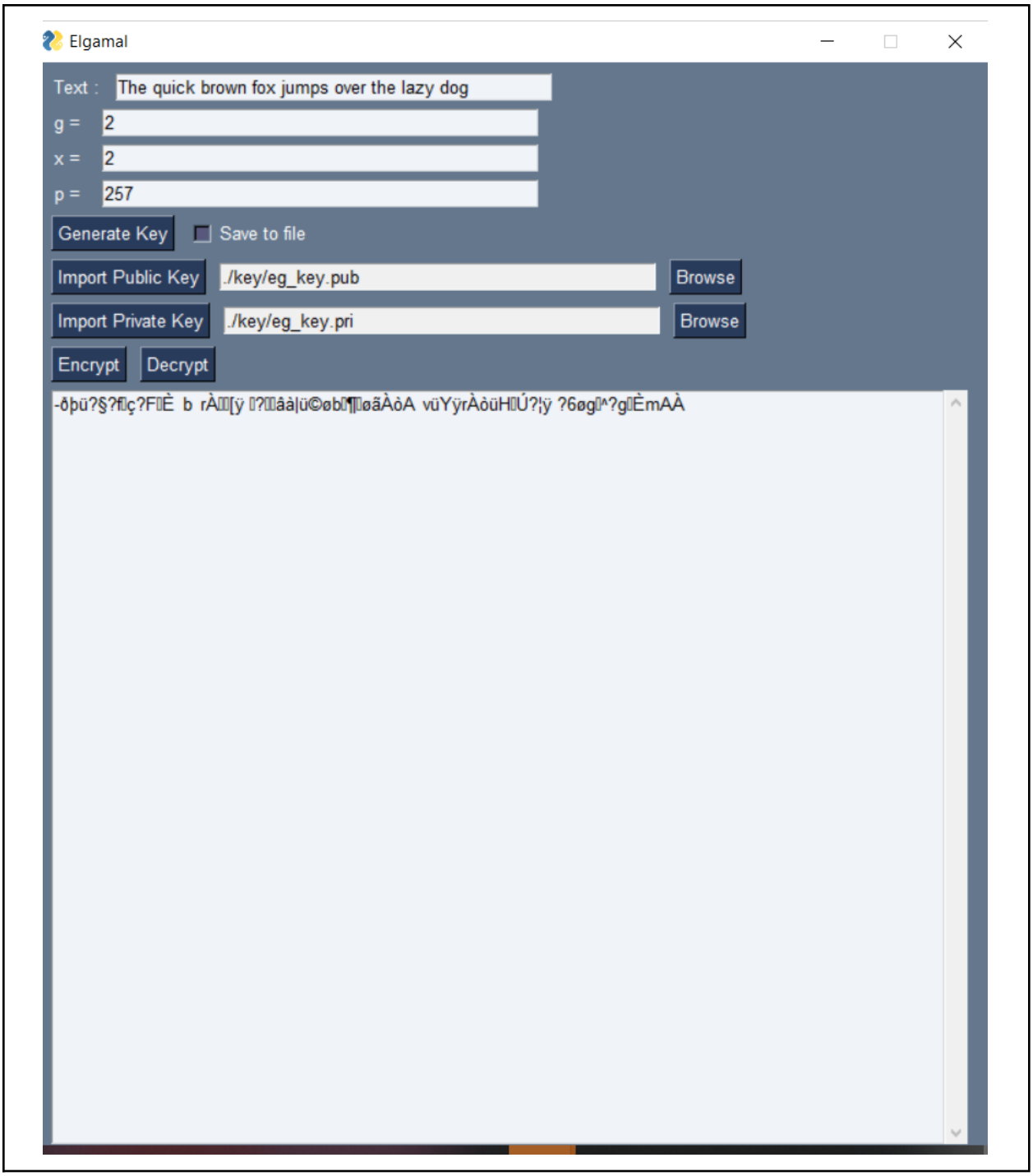
4.2. ElGamal

4.2.1. Uji Coba 1

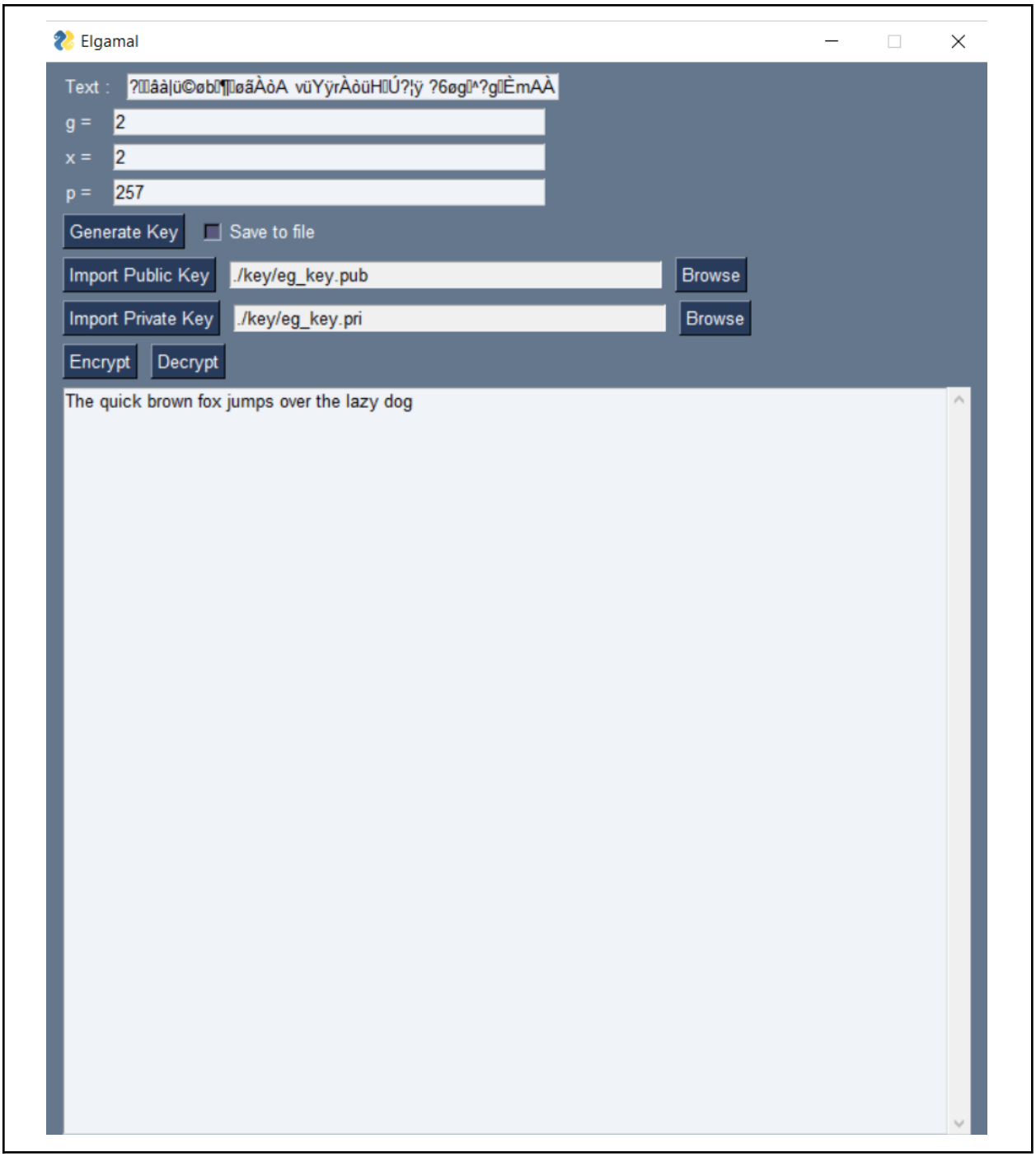
Plaintext : “The quick brown fox jumps over the lazy dog”
Generate Key :

Public Key	Private Key
g=2 y=4 p=257	x=2 p=257

Ciphertext : “ð—þ- ü
œ?§?fç?FÈ
”



Plaintext : “The quick brown fox jumps over the lazy dog”



4.2.2. Uji Coba 2

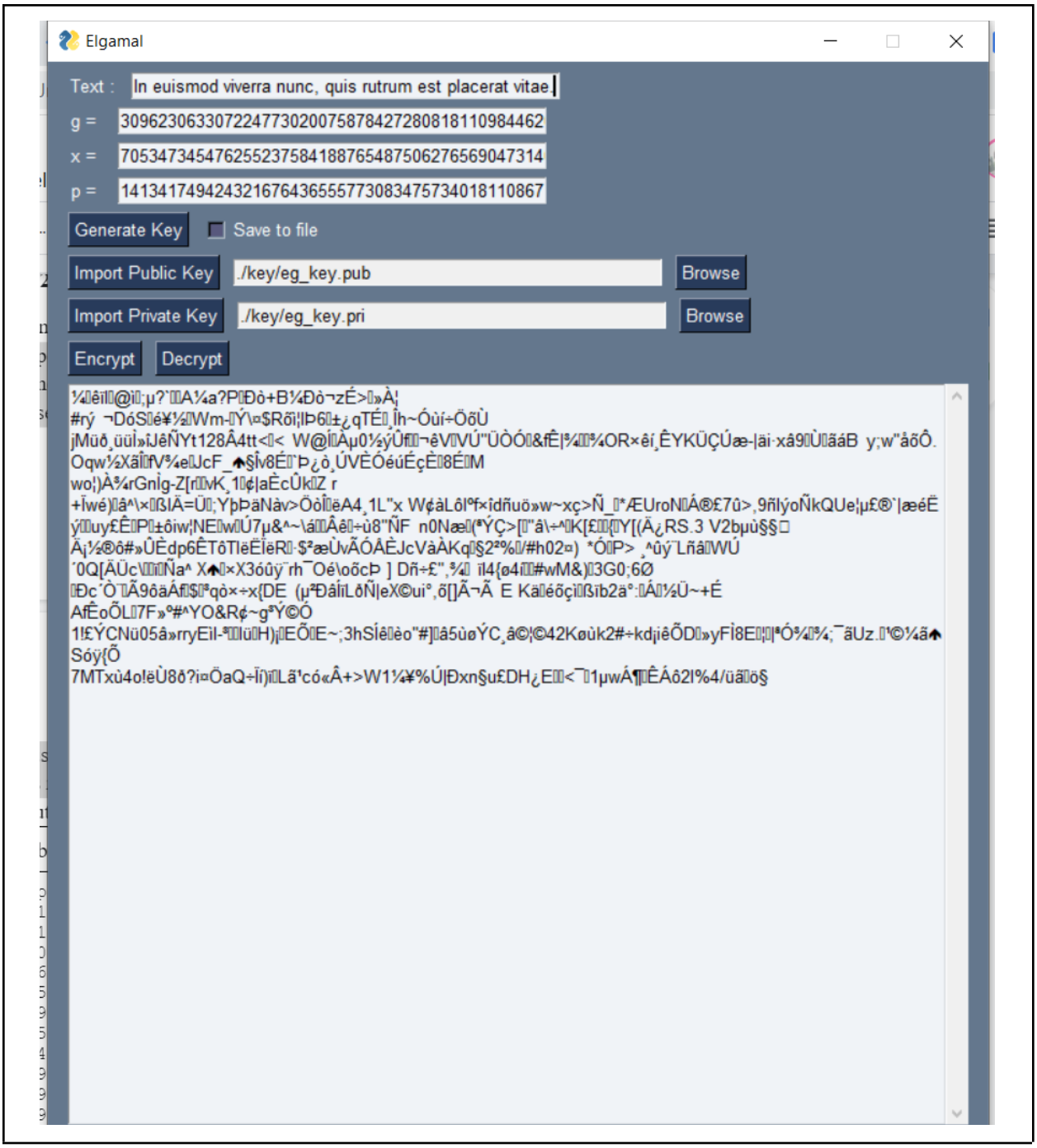
Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur

lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”

Input Key :

Public Key	Private Key
{"p": 1413417494243216764365557730834757340181108671797506650121861601831627178015087849031276434821249092215658079104461937170218004525824954395919733920005608845154552086259408045103339196085950347144496187898974434543647253688557297677860698406856272205703104588304267777782105986887680808347733139189093749363633, "g": 3096230633072247730200758784272808181109844620693738954771669530930355953787488220250047634308553636924505965934782951741001080544605673953657981430501499742000893048642821889415590052535587094713358156042081112673280448131745262863414165856145607566670357572533306067959514763073878864332148236556688018455 3, "y": 1358569231022793779583236105597667585891979189593742318716323713123866182994899156362767796341845704698656508358543378325073943333002736956064223699234008816860941874962010884784852511130785613953971899755706606617204363670151218634728274218011108703825508007656513053437669129556893188215371606086629230784 24}	{"p": 1413417494243216764365557730834757340181108671797506650121861601831627178015087849031276434821249092215658079104461937170218004525824954395919733920005608845154552086259408045103339196085950347144496187898974434543647253688557297677860698406856272205703104588304267777782105986887680808347733139189093749363633, "x": 7053473454762552375841887654875062765690473140718192155317265999617496796305915974481725391371667695697095178507212331168871526875565341608493737666950177970121079570715303361065546297819530666813624965573208341668481688988759029472364723016338230814307643137098455757397413777578283764193059662119865324 6333}

Ciphertext : “¼êïlŠ@”ì;µ?`™A¼a?PÐò+B¼Ðò¬zÉ>»À¦
#†rý¬DóSé¥›½WmÝ\, ¤$Rõì„¦, lÞ6±¿qTÉ¸Îh~Óù”í÷ÖõÙ
jMüðŒ¸ü—üÌ»îJêÑYt128Â4tt<- <†W@ÍÀµ0½ý…Ûf¬êV, VÚ"ÜÒÓ&fÊ…|¾¾OR›×êí¸ŸÊYKÜ
ÇÚæ|„äi’·xâ9ÙãáBy;w"åõÔ.O˜qw½X˜ãÎfV¾eJcF_§Îv8, ˜É, `Þ¿ò¸ÚV›ÈÓéúÉçÈ8ÉM
wo¦)À“¾rGnÌg-Z[r…vK¸1„¢Ÿ-|aÈcÛkZ™ ›r
+Ïwé)â^›\×ßIÄ=Ü;YþÞäNàvŸ>ÖòÎëA4¸1L"Ÿx
W¢àLôI‡ƒºf×îdñuö, »w~xç>Ñ_*ÆU—roNÁ®£7˜û>,9ñ“lýoÑkQUe¦µ£—®`|æéËý, uy‹ £ÊƒP±ôi
w¦NEwÚ…7µ&^~Š\áÂê÷ù8"ÑFn0Næ”(ªÝ›Ç>["â˜˜\÷^™K[”£{—Y[(Ä¿RS.3
’V2bµù‡§§Ä“¡½®ô—#»Û…Èdp6ÊŠTôTIëËÏ, ëR·$²æÙ-vÃÓÅÈJcVà‰ÀKq§2²%/- #h02¤)“
”



Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”


Elgamal
Text :
>W1¼¥%Ú|Ðxn§u£DH¿E<¯1µwÁ¶ÊÁô2l%4/üãö§
g =
309623063307224773020075878427280818110984462
x =
705347345476255237584188765487506276569047314
p =
141341749424321676436555773083475734018110867
Generate Key
Save to file
Import Public Key
./key/eg_key.pub
Browse
Import Private Key
./key/eg_key.pri
Browse
Encrypt
Decrypt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.

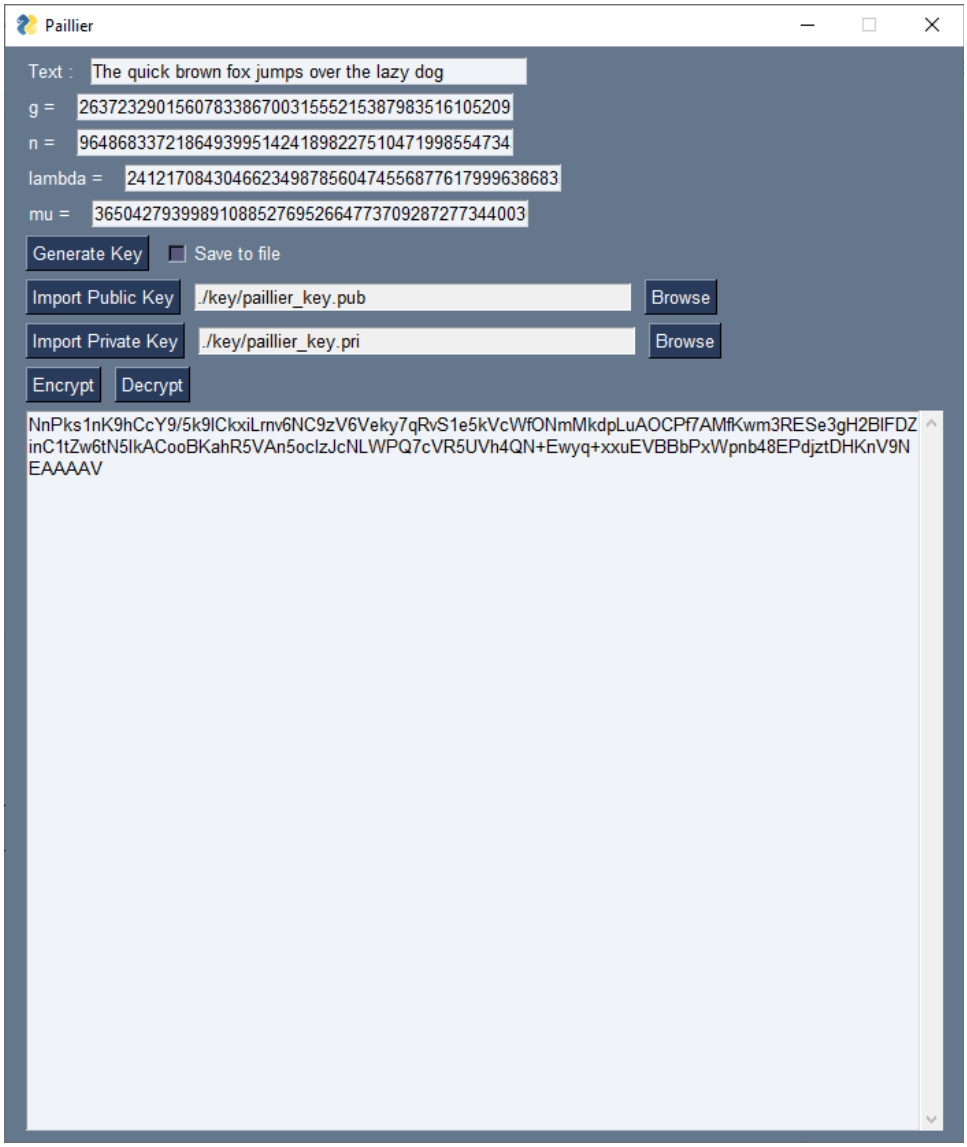
4.3. Paillier

4.3.1. Uji Coba 1

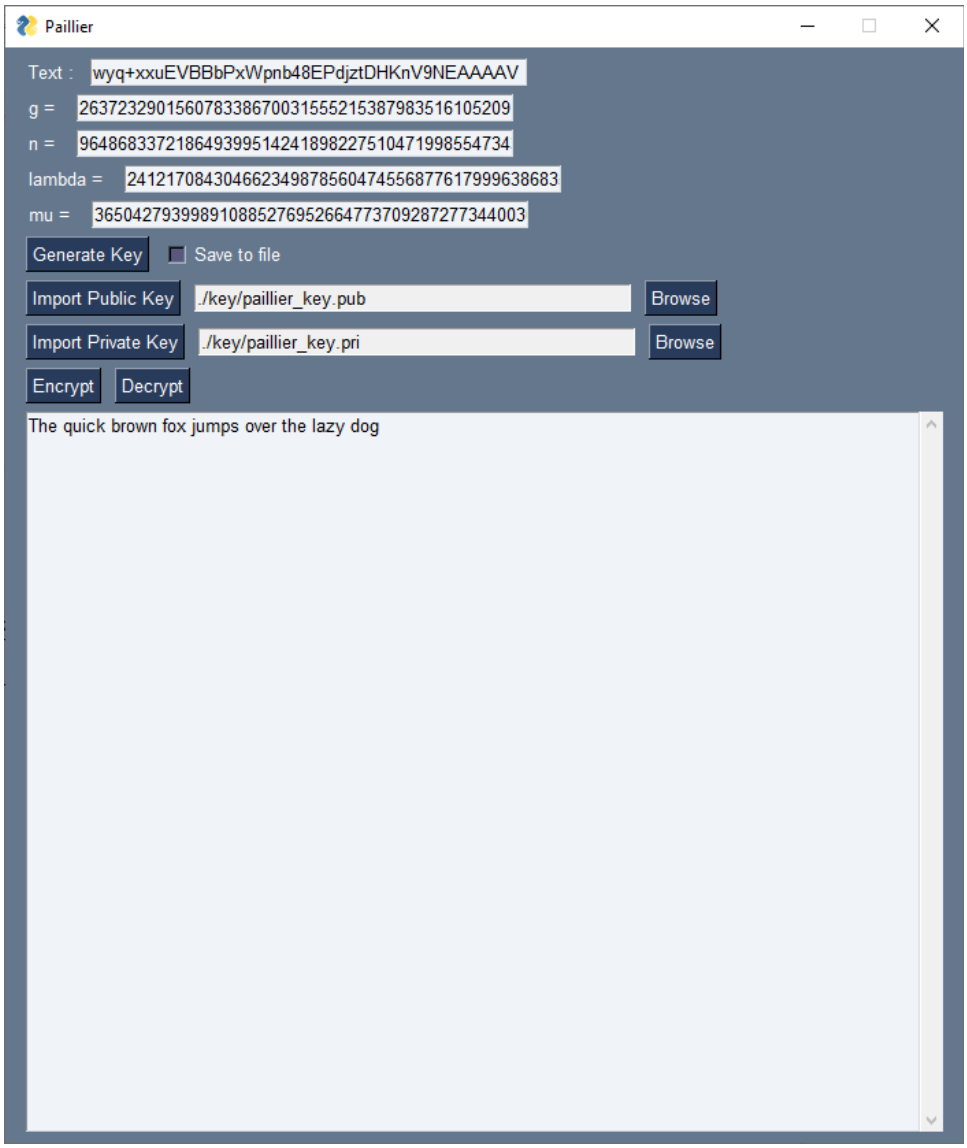
Plaintext : “The quick brown fox jumps over the lazy dog”
Generate Key :

Public Key	Private Key
g= 2637232901560783386700315552153879835161052091812934410158290604022099702525649333323969451914632325929944829691059536769750282686439315210195195147336169490877138661245220699818931486082001539238107646277622622577634312714964556803408454713571336389339771190637626924271579561001838988481240559106844260 7682 n= 9648683372186493995142418982275104719985547343851241351225400717404116216495184153068705262767957726680762757005651831511775980293275594838674217482834237	g= 2637232901560783386700315552153879835161052091812934410158290604022099702525649333323969451914632325929944829691059536769750282686439315210195195147336169490877138661245220699818931486082001539238107646277622622577634312714964556803408454713571336389339771190637626924271579561001838988481240559106844260 7682 n= 9648683372186493995142418982275104719985547343851241351225400717404116216495184153068705262767957726680762757005651831511775980293275594838674217482834237 λ= 2412170843046623498785604745568776179996386835962810337806350179351029054123746459596146390029448789666598180495099001820292543899837647820245499465972944 μ= 3650427939989108852769526647737092872773440030939183348479286161126911437041837095117654193084232217374091022580577248102491129029879794231684166952526099

Ciphertext :
“NnPks1nK9hCcY9/5k9lCkxiLrnv6NC9zV6Veky7qRvS1e5kVcWfONmMkdpLuAOCPf7AMfKwm3RESe3gH2BlFDZinC1tZw6tN5lkACooBKahR5VAn5ocIzJcNLWPQ7cVR5UVh4QN+Ewyq+xxuEVBBbPxWpnb48EPdjztDHKnV9NEAAAAV”



Plaintext : “The quick brown fox jumps over the lazy dog”



4.3.2. Uji Coba 2

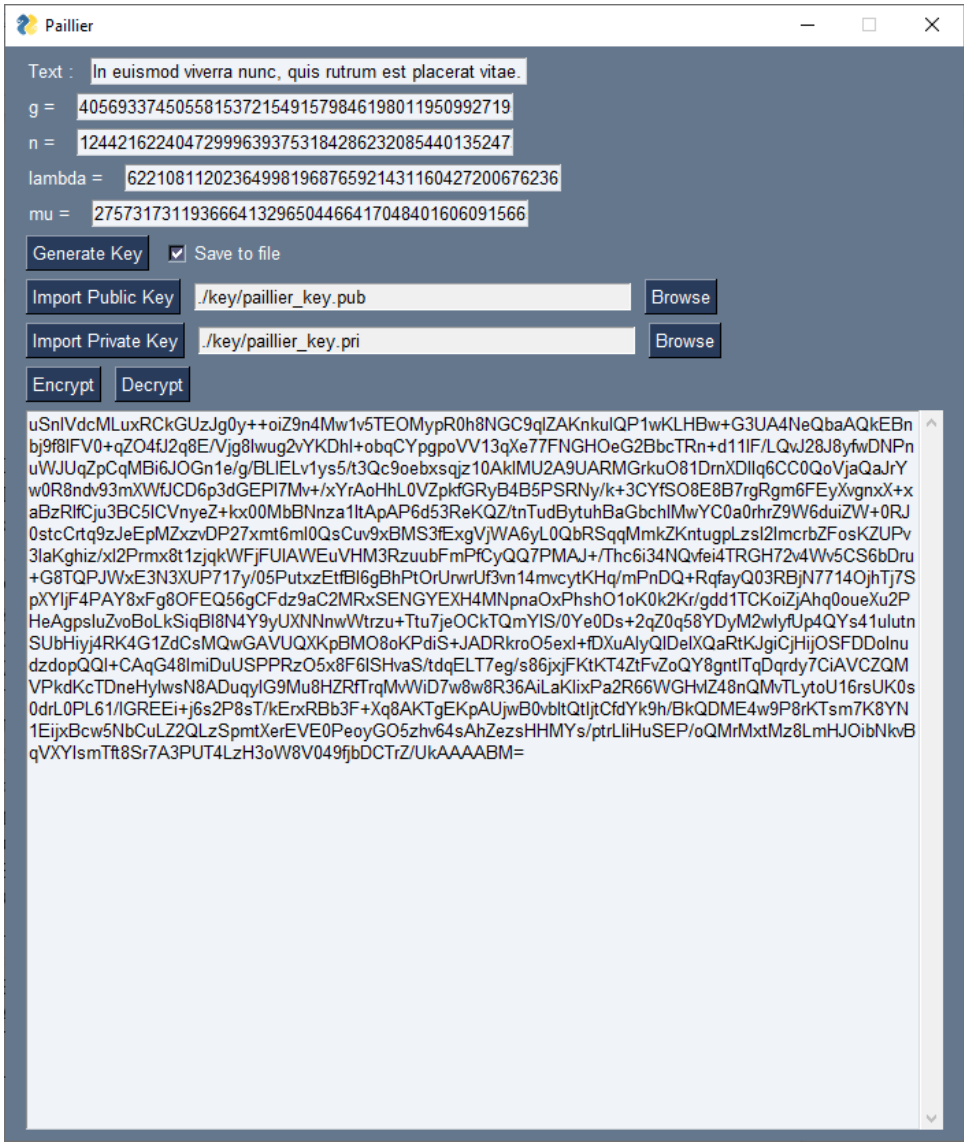
Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”

Input Key :

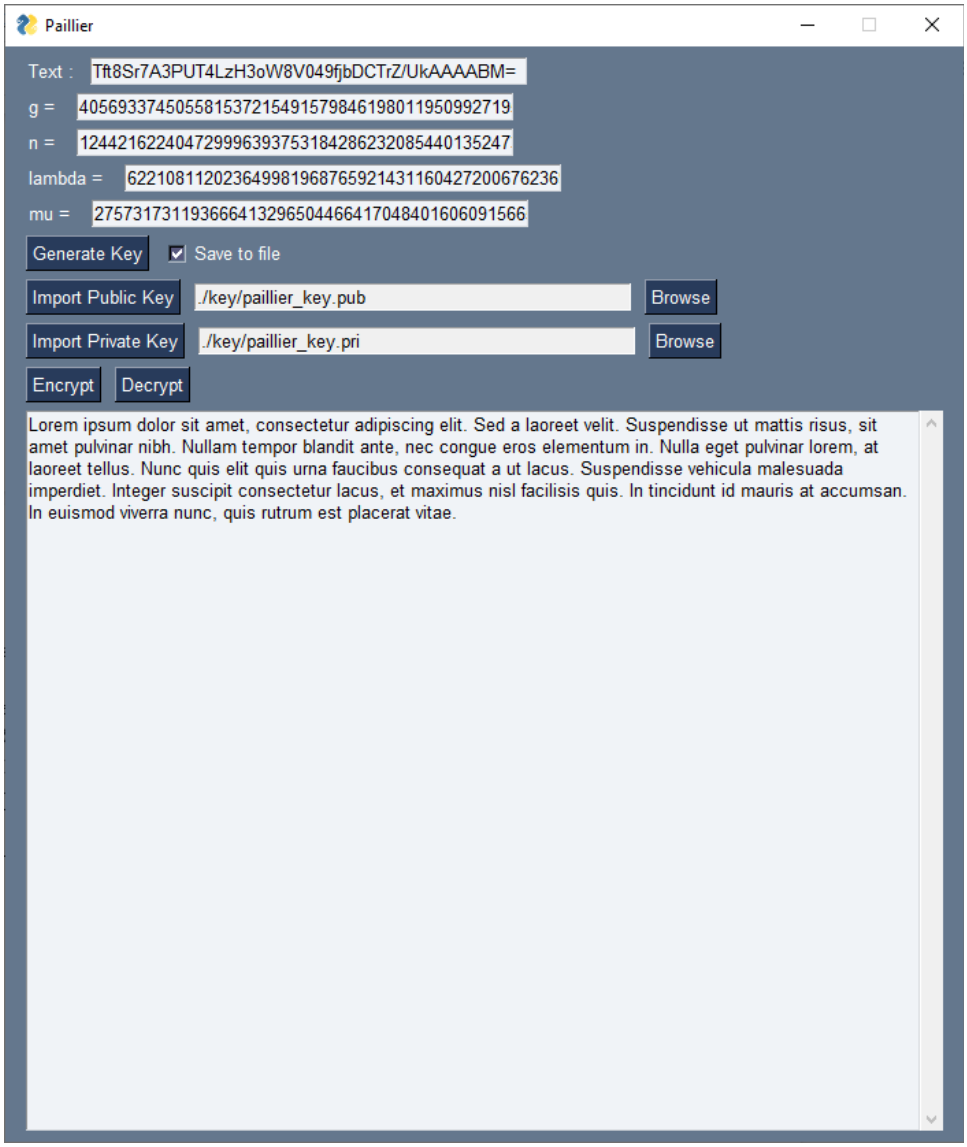
Public Key	Private Key
g= 40569337450558153721549157984619801195 09927195171791730415078531560573151621 00096730405974319783323282966939313719 85064076771475287666589229541704062007 13302778507255856560539895713636937506 76378014811144162591673224824279052367 92397542877225841628740987978591575823 20470616127241623366980745808445711536 7175 n= 12442162240472999639375318428623208544 01352473901214344689737360764278548901 50391028343889208329518823276155571079 24970321984116174717165602018158466006 191	g= 40569337450558153721549157984619801195 09927195171791730415078531560573151621 00096730405974319783323282966939313719 85064076771475287666589229541704062007 13302778507255856560539895713636937506 76378014811144162591673224824279052367 92397542877225841628740987978591575823 20470616127241623366980745808445711536 7175 n= 12442162240472999639375318428623208544 01352473901214344689737360764278548901 50391028343889208329518823276155571079 24970321984116174717165602018158466006 191 λ= 62210811202364998196876592143116042720 06762369506071723448686803821392744507 40793112710593464417412438207606127029 73452149995696544816244927635782428696 36 μ= 27573173119366641329650446641704840160 60915665608311373495587131891566949089 46445239192581609877193234642470966588 70687427941771377124963799143628832026 19

Ciphertext :

“uSnlVdcMLuxRCkGUzJg0y++oiZ9n4Mw1v5TEOMypR0h8NGC9qlZAKnkulQP1wKLHBw+
G3UA4NeQbaAQkEBnbj9f8lFV0+qZO4fJ2q8E/Vjg8lwug2vYKDhI+obqCYpgpoVV13qXe77F
NGHOeG2BbcTRn+d11IF/LQvJ28J8yfwDNPnuWJUqZpCqMBi6JOGn1e/g/BLlELv1ys5/t3Qc
9oebxsqjz10AklMU2A9UARMGrkuO81DrnXDlIq6CC0QoVjaQaJrYw0R8ndv93mXWfJCD6p
3dGEPI7Mv+/xYrAoHhL0VZpkfGRyB4B5PSRNy/k+3CYfSO8E8B7rgRgm6FEyXvgnxX+xa
BzRIfCju3BC5ICVnyeZ+kx00MbBNnza1ltApAP6d53ReKQZ/tnTudBytuhBaGbchIMwYC0a0r
hrZ9W6duiZW+0RJ0stcCrtq9zJeEpMZxzvDP27xmt6mI0QsCuv9xBMS3fExgVjWA6yL0QbRS
qqMmkZKntugpLzsl2lmcrbZFosKZUPv3laKghiz/xl2Prmx8t1zjqkWFjFUlAWEuVHM3RzuubF
mPfCyQQ7PMAJ+/Thc6i34NQvfei4TRGH72v4Wv5CS6bDru+G8TQPJWxE3N3XUP717y/05
PutxzEtfBI6gBhPtOrUrwrUf3vn14mvcytKHq/mPnDQ+RqfayQ03RBjN7714OjhTj7SpXYIjF4P
AY8xFg8OFEQ56gCFdz9aC2MRxSENGYEXH4MNpnaOxPhshO1oK0k2Kr/gdd1TCKoiZjAh
q0oueXu2PHeAgpsIuZvoBoLkSiqBl8N4Y9yUXNNnwWtrzu+Ttu7jeOCkTQmYIS/0Ye0Ds+2q
Z0q58YDyM2wIyfUp4QYs41ulutnSUbHiyj4RK4G1ZdCsMQwGAVUQXKpBMO8oKPdiS+J
ADRkroO5exI+fDXuAlyQIDelXQaRtKJgiCjHijOSFDDoInudzdopQQl+CAqG48ImiDuUSPPR
zO5x8F6lSHvaS/tdqELT7eg/s86jxjFKtKT4ZtFvZoQY8gntITqDqrdy7CiAVCZQMVPkdKcTDn
eHylwsN8ADuqyIG9Mu8HZRfTrqMvWiD7w8w8R36AiLaKIixPa2R66WGHvIZ48nQMvTLyt
oU16rsUK0s0drL0PL61/lGREEi+j6s2P8sT/kErxRBb3F+Xq8AKTgEKpAUjwB0vbltQtljtCfdY
k9h/BkQDME4w9P8rKTsm7K8YN1EijxBcw5NbCuLZ2QLzSpmtXerEVE0PeoyGO5zhv64sA
hZezsHHMYs/ptrLIiHuSEP/oQMrMxtMz8LmHJOibNkvBqVXYIsmTft8Sr7A3PUT4LzH3oW
8V049fjbDCTrZ/UkAAAABM=”


Paillier
Text :
In euismod viverra nunc, quis rutrum est placerat vitae.
g =
405693374505581537215491579846198011950992719
n =
124421622404729996393753184286232085440135247
lambda =
622108112023649981968765921431160427200676236
mu =
275731731193666413296504466417048401606091566
Generate Key
Save to file
Import Public Key
./key/paillier_key.pub
Browse
Import Private Key
./key/paillier_key.pri
Browse
Encrypt
Decrypt
uSnlVdcMLuxRCkGUzJg0y++oiZ9n4Mw1v5TEOMypR0h8NGC9qlZAKnkuIQP1wKLHBw+G3UA4NeQbaAQkEBn
bj9f8lFV0+qZO4fJ2q8E/Vjg8lwug2vYKDhl+obqCYpgpoVV13qXe77FNGHOeG2BbcTRn+d11lF/LQvJ28J8yfwDNPn
uWJUqZpCqMBi6JOGn1e/g/BLlELv1ys5/t3Qc9oebxsqjz10AklMU2A9UARMGrkuO81DrnXDlIq6CC0QoVjaQaJrY
w0R8ndv93mXWfJCD6p3dGEPl7Mv+/xYrAoHhL0VZpkfGRyB4B5PSRNy/k+3CYfSO8E8B7rgRgm6FEyXvgnxX+x
aBzRlfCju3BC5ICVnyeZ+kx00MbBNnza1ltApAP6d53ReKQZ/tnTudBytuhBaGbchlMwYC0a0rhrZ9W6duiZW+0RJ
0stcCrtq9zJeEpMZxzvDP27xmt6ml0QsCuv9xBMS3fExgVjWA6yL0QbRSqqMmkZKntugpLzsI2ImcrbZFosKZUPv
3laKghiz/xl2Prmx8t1zjqkWFjFUlAWEuVHM3RzuubFmPfCyQQ7PMAJ+/Thc6i34NQvfei4TRGH72v4Wv5CS6bDru
+G8TQPJWxE3N3XUP717y/05PutxzEtfBl6gBhPtOrUrwrUf3vn14mvcytKHq/mPnDQ+RqfayQ03RBjN7714OjhTj7S
pXYljF4PAY8xFg8OFEQ56gCFdz9aC2MRxSENGYEXH4MNpnaOxPhshO1oK0k2Kr/gdd1TCKoiZjAhq0oueXu2P
HeAgpsIuZvoBoLkSiqBl8N4Y9yUXNNnwWtrzu+Ttu7jeOCkTQmYIS/0Ye0Ds+2qZ0q58YDyM2wlyfUp4QYs41ulutn
SUbHiyj4RK4G1ZdCsMQwGAVUQXKpBMO8oKPdiS+JADRkroO5exl+fDXuAlyQIDelXQaRtKJgiCjHijOSFDDolnu
dzdopQQl+CAqG48ImiDuUSPPRzO5x8F6lSHvaS/tdqELT7eg/s86jxjFKtKT4ZtFvZoQY8gntITqDqrdy7CiAVCZQM
VPkdKcTDneHylwsN8ADuqylG9Mu8HZRfTrqMvWiD7w8w8R36AiLaKlixPa2R66WGHMZ48nQMvTLytoU16rsUK0s
0drL0PL61/lGREEi+j6s2P8sT/kErxRBb3F+Xq8AKTgEKpAUjwB0vbltQtljtCfdYk9h/BkQDME4w9P8rKTsm7K8YN
1EijxBcw5NbCuLZ2QLzSpmtXerEVE0PeoyGO5zhv64sAhZezsHHMYs/ptrLliHuSEP/oQMrMxtMz8LmHJOibNkvB
qVXYIsmTft8Sr7A3PUT4LzH3oW8V049fjbDCTrZ/UkAAAABM=

Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in. Nulla eget pulvinar lorem, at laoreet tellus. Nunc quis elit quis urna faucibus consequat a ut lacus. Suspendisse vehicula malesuada imperdiet. Integer suscipit consectetur lacus, et maximus nisl facilisis quis. In tincidunt id mauris at accumsan. In euismod viverra nunc, quis rutrum est placerat vitae.”



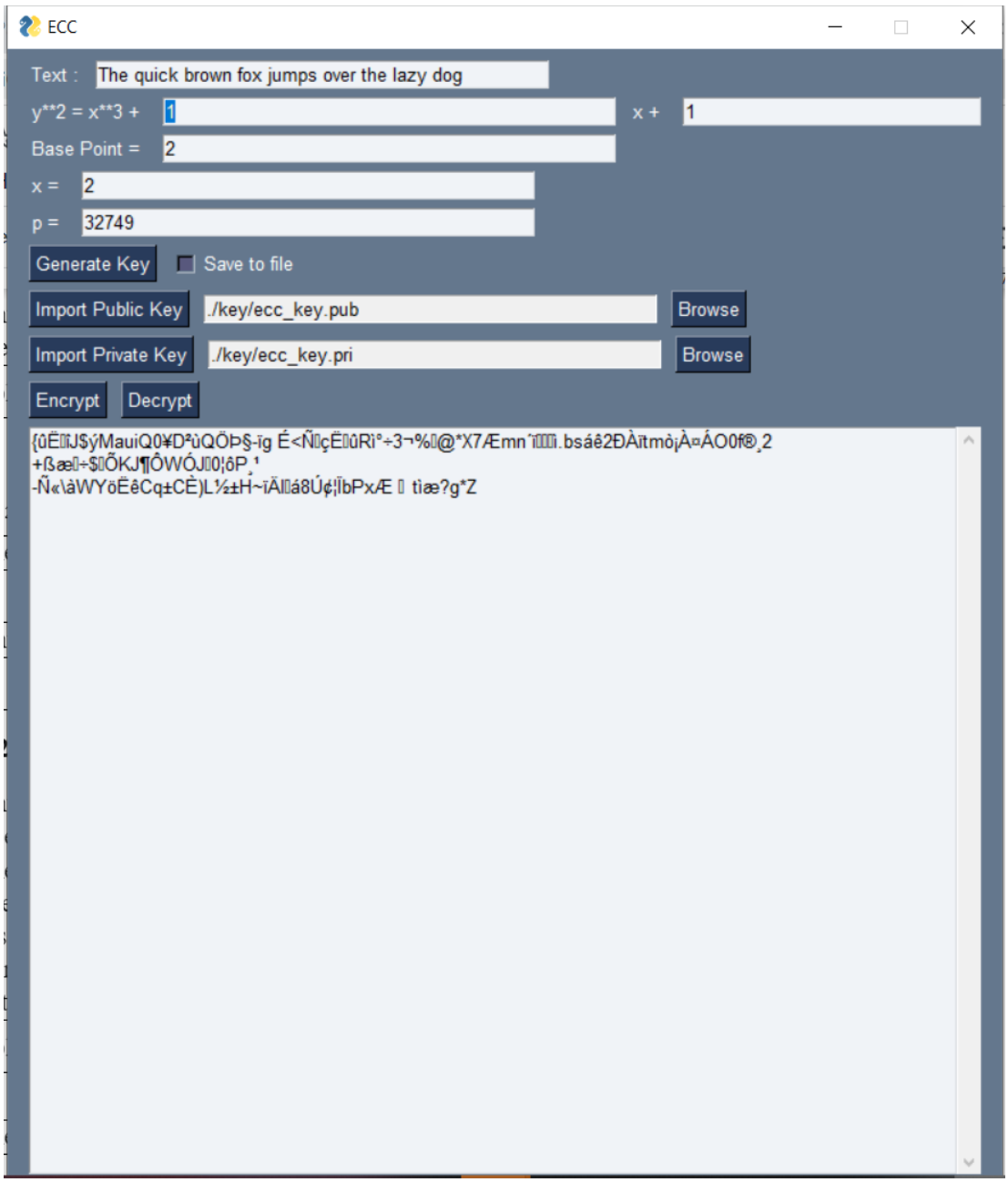
4.4. ECC

4.4.1. Uji Coba 1

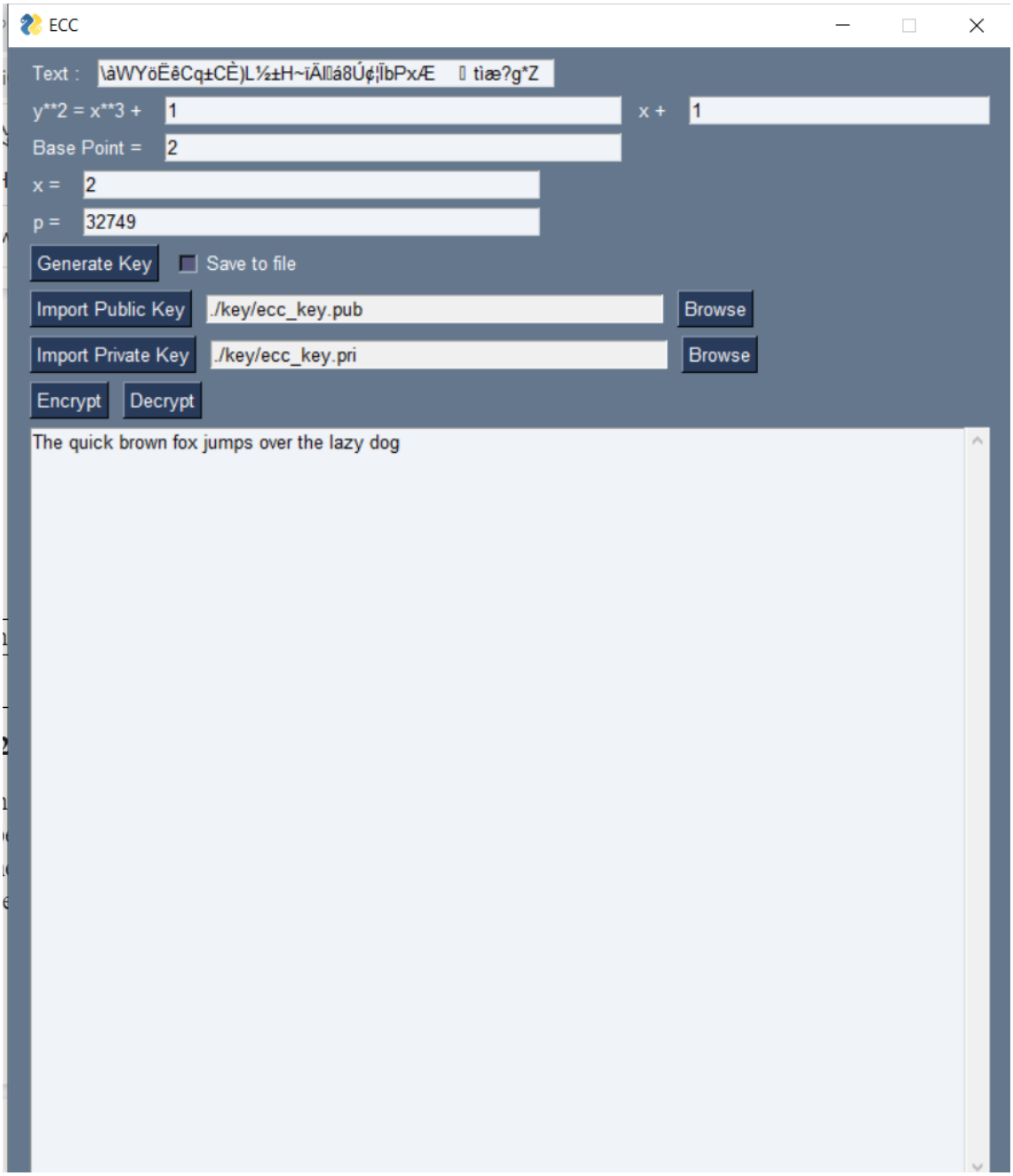
Plaintext : “The quick brown fox jumps over the lazy dog”
Generate Key :

Public Key	Private Key
a=1 b=1 g=2 p=32749	x=2 p=32749

Ciphertext : “˜{ûË”îJ’$ý”MauiQ0¥D²ùQÖÞ§-ïg
É<ÑçŠËûRì°÷3¬%“@*X7Æmn´ï™ì.bsáêš2ÐÀïtmò¡À¤ÁO0f®¸2
+ßæ†÷$…ÕKJ¶ÔWÓJ0¦ôP¸¹œ
Ñ«\àWYöËêCq±CÈ)L½±H~ïÄ˜lá8Ú˜¢¦ÏbPxÆ ‡tìæ?g*Z
”



Plaintext : “The quick brown fox jumps over the lazy dog”



4.4.2. Uji Coba 2

Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in.”

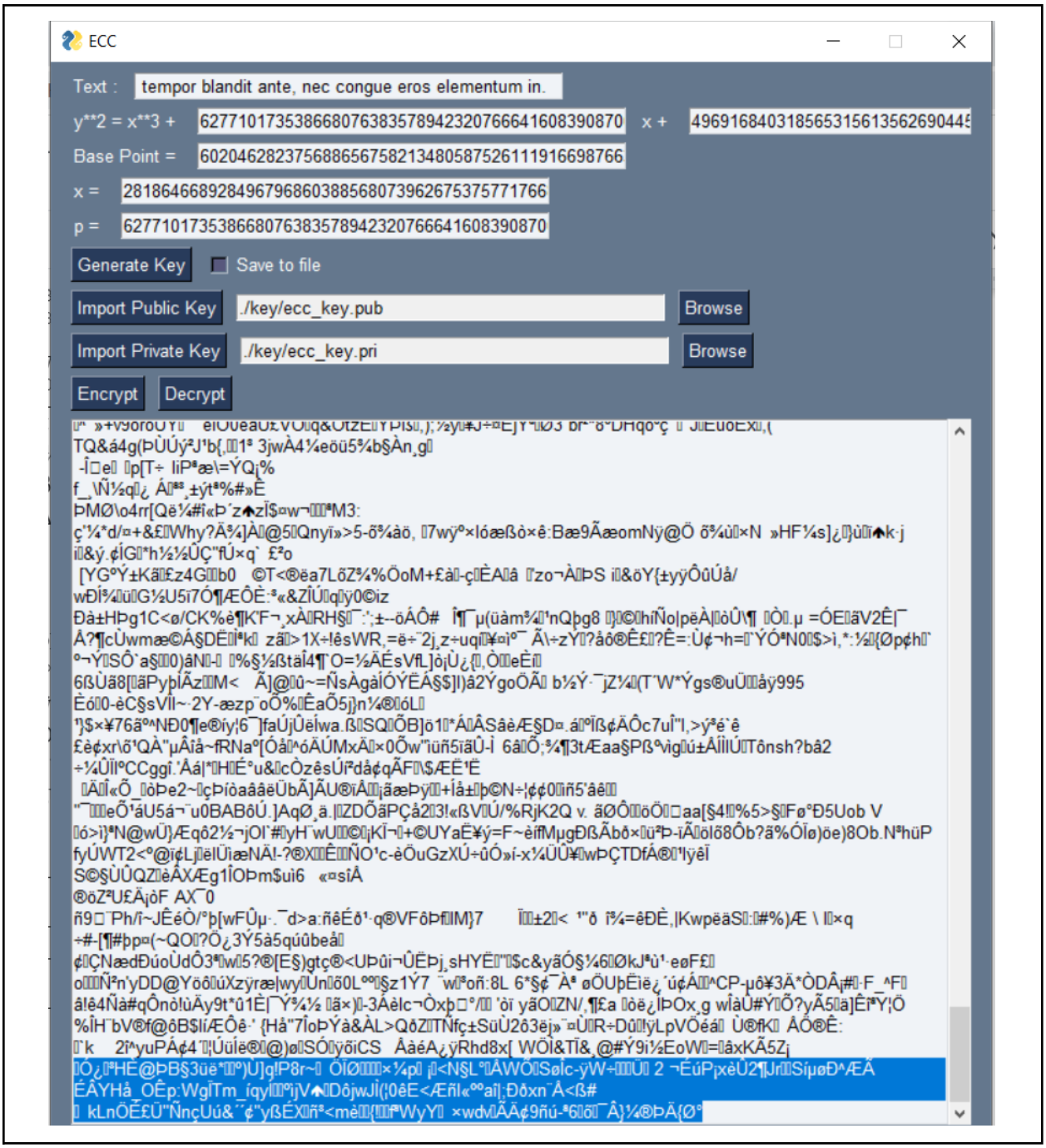
Input Key :

Public Key	Private Key

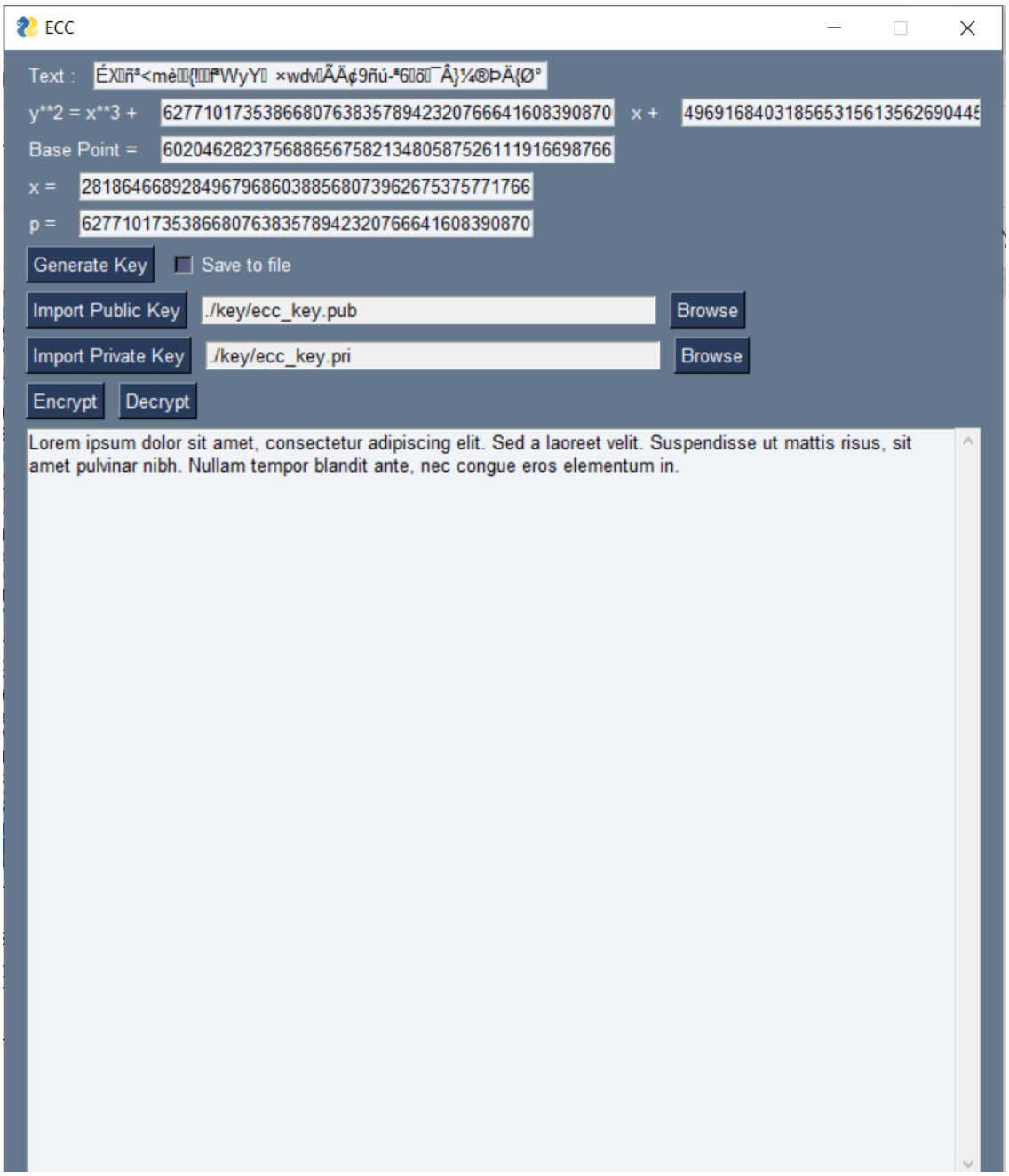
a= 62771017353866807638357894232076664160 83908700390324961276 b= 49691684031856531561356269044583696279 46108939723516709469 g= [60204628237568865675821348058752611191 669876636884684818, 17405033229362203140485755228021941036 4023488927386650641] p= 62771017353866807638357894232076664160 83908700390324961279	x= 28186466892849679686038856807396267537 577176687436853369 p= 62771017353866807638357894232076664160 83908700390324961279

Ciphertext : Panjang, karena 1 karakter dipetakan ke 4*(banyak byte dari p) karakter, yakni

“&gæä8B¼Ù©¬ö® FõE‘ß‹ ¸ÙÌc6øÀgÏÌ]¡^pEfÉÒC
emvC$cxß §8Qó)“QND^'jÍ8j™±#“ µDÃ8c(æzœùÝÌñLkßãB…>¼©a¿*
=?Q«Ö¾Áé”T´âÕUL[ÚM±Git²À-™íøÿ¯ ´H»é*·og^GµFtioI#Ò|…ÉE?}mµ†
…
…
...
Ó¿³HÈ@ÞB§3üë†*º)Ù]q!P8, “…r~Š›ÖÏØ×¼p ¡
˜<N§L„ °Å
WÕSøÎc-ÿ˜W÷Ü 2 ¬ÉúP™¡xèÛ2¶JrSíµøÐ^ÆÃ
ÉÂYHå_OÊp:WgÏTœm_íqyÍºijVDôjwJÌ(¦0êE<Æñl«ººaî
|;Ððxn¨Å<ß#
kLnÖÉ£Ü"ÑnçUúƒ&´´¢"yßÉ”Xñ³Œ<mè{!šfªWy’Y„×ŸwdvÃÄ¢9ñúª6õ¯Â}¼®ÞÄ…{Ø°
”



Plaintext : “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in.”


ECC
Text :
y**2 = x**3 +
627710173538668076383578942320766641608390870
x +
Base Point =
602046282375688656758213480587526111916698766
x =
281864668928496796860388568073962675375771766
p =
627710173538668076383578942320766641608390870
Generate Key
Save to file
Import Public Key
./key/ecc_key.pub
Browse
Import Private Key
./key/ecc_key.pri
Browse
Encrypt
Decrypt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed a laoreet velit. Suspendisse ut mattis risus, sit amet pulvinar nibh. Nullam tempor blandit ante, nec congue eros elementum in.

Bab 5. Pranala

Git Repository : BeforeLast/Asymmetric-Cryptography (github.com)
Google Drive : Folder - Google Drive

Bab 6. Pembagian Tugas

NIM	Tugas
13519044	ElGamal, ECC, GUI
13519074	RSA, Paillier, GUI