

# Maturita2020

## *Cifratura-Decifratura di file in RSA-AES*

### I – Introduzione

Lo scopo di questa relazione è descrivere gli strumenti, le librerie di terze parti e l'implementazione algoritmica con i quali è stato sviluppato un applicativo per la cifratura e decifratura di file (di qualunque formato) e il salvataggio degli stessi nella working directory.

Per farlo, saranno anche spiegate le problematiche riscontrate durante le varie fasi di progetto e le strategie che si sono utilizzate per risolverle, fornendo esempi di codice commentato e vari screenshots scattati durante lo sviluppo.

### II – Fase 1

#### Design dell'interfaccia grafica (GUI)

La prima fase di sviluppo dell'applicativo ha riguardato la progettazione di un'interfaccia grafica attraverso cui l'utente potesse interagire con le funzionalità presenti.

I criteri fondamentali su cui si è deciso di basare la realizzazione di questa interfaccia sono stati:

- La semplicità di utilizzo per l'utente finale
- La semplicità di sviluppo per il designer-programmatore

Si è deciso quindi di puntare su un qualcosa che fosse veloce da progettare e allo stesso tempo funzionale alle esigenze dell'utente finale di media esperienza (target a cui potrebbe interessare un applicativo simile), quindi non troppo complesso, con funzionalità o widget inutili, almeno in una prima release.

Un approccio quindi minimalista, se vogliamo, molto pragmatico, che ha cercato di concentrarsi sull'obiettivo finale (avere un applicativo utilizzabile da interfaccia grafica) evitando gli sprechi o i dettagli estetici che avrebbero richiesto un maggior impiego di tempo e risorse in una fase ritenuta non fondamentale per la realizzazione del progetto.

Dati i criteri definiti, i tempi e gli obiettivi, si è quindi deciso di puntare su uno strumento come PyQt5, perfetto per il nostro scopo in quanto indicato per realizzare velocemente interfacce grafiche anche di notevole complessità pur

risparmiandoci gli sforzi che invece avremmo sostenuto puntando su alternative anche magari di maggior spessore, come C#, .NET o Java.

PyQt5 è una libreria Python contenente i bindings per l'accesso e l'interoperabilità tra il linguaggio e il GUI toolkit multiplatforma Qt.

Documentazione: <https://doc.bccnsoft.com/docs/PyQt5>

Qt, come già detto, è una libreria grafica per lo sviluppo di interfacce tramite widget, drag and drop e interfaccia grafica.

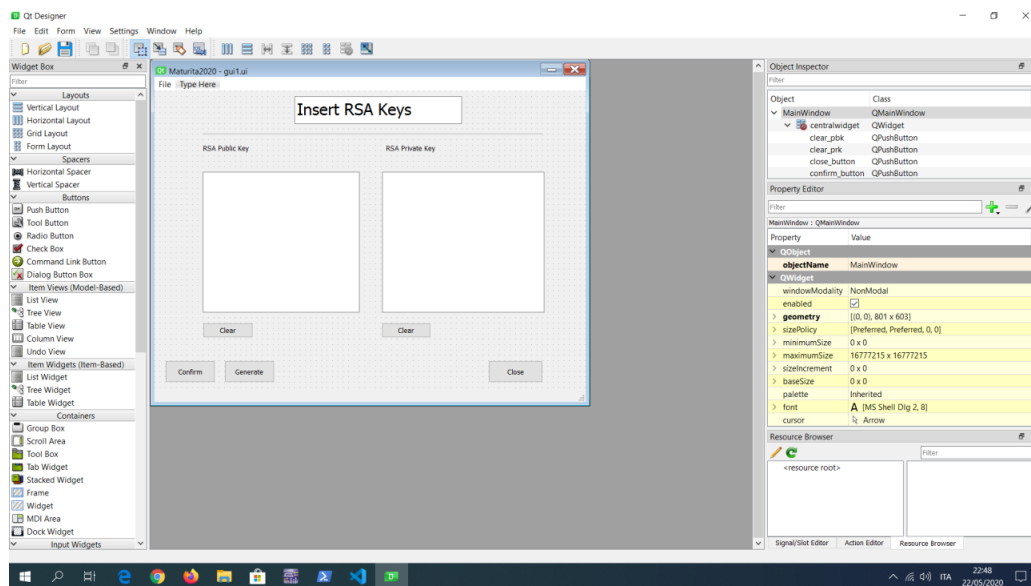
Documentazione: <https://wiki.qt.io/Main>

Per non dilungarsi troppo nella spiegazione troppo dettagliata di entrambi gli strumenti, che richiederebbero entrambi una guida a sé, si rimanda alla documentazione ufficiale per ulteriori approfondimenti.

E' importante però precisare che la scelta di Python come linguaggio di sviluppo in questa fase si rivela senza dubbio adatta a soddisfare i criteri di semplicità e velocità che abbiamo stabilito prima.

Il potente paradigma Object Oriented del linguaggio infatti si presta perfettamente a uno sviluppo veloce, immediato, con un notevole risparmio di righe di codice, a differenza per esempio di alternative come C++, C# o Java.

Di seguito, si allegano alcuni screenshots scattati durante questa fase di sviluppo:



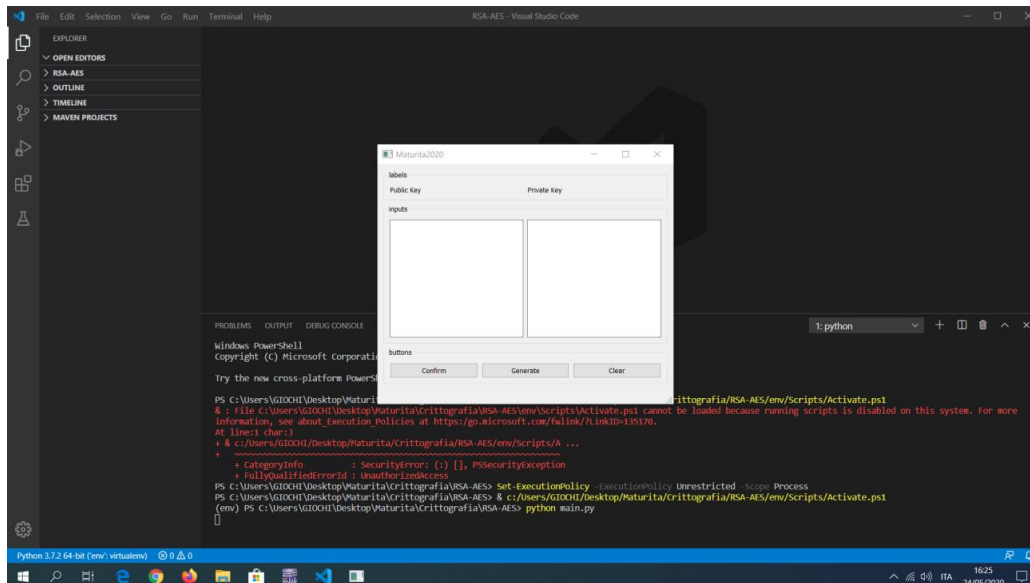
Installando la libreria PyQt5\_tools si può avere accesso al GUI designer Qt Designer, una versione di Qt progettata per operare con la libreria PyQt5.

Documentazione:

<https://doc.qt.io/qt-5/qtdesigner-manual.html>

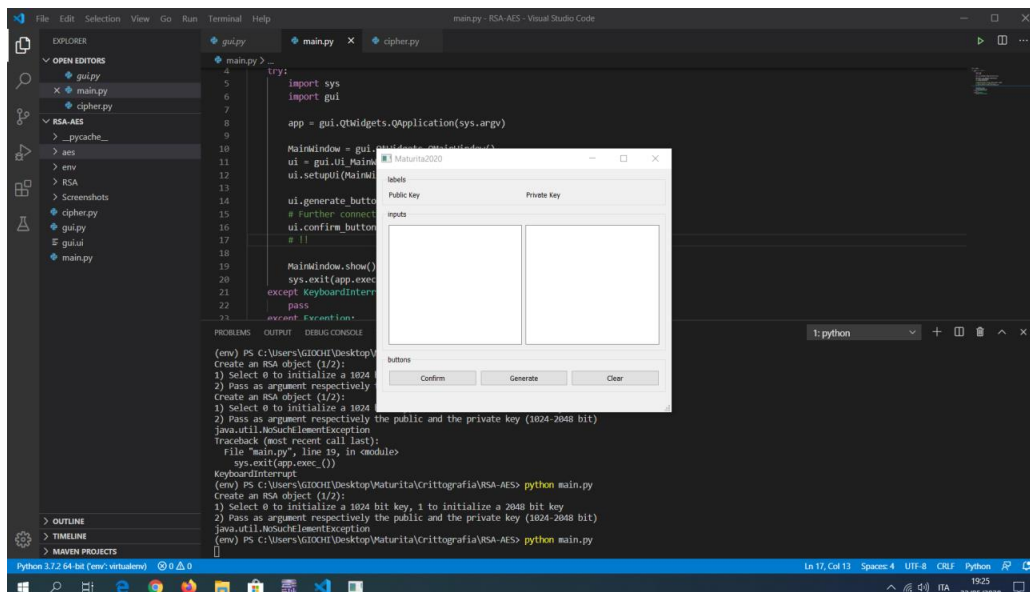
Per installare la libreria PyQt5\_tools si può utilizzare normalmente il Python Package Manager pip, con il comando:

```
>> pip install pyqt5-tools
```



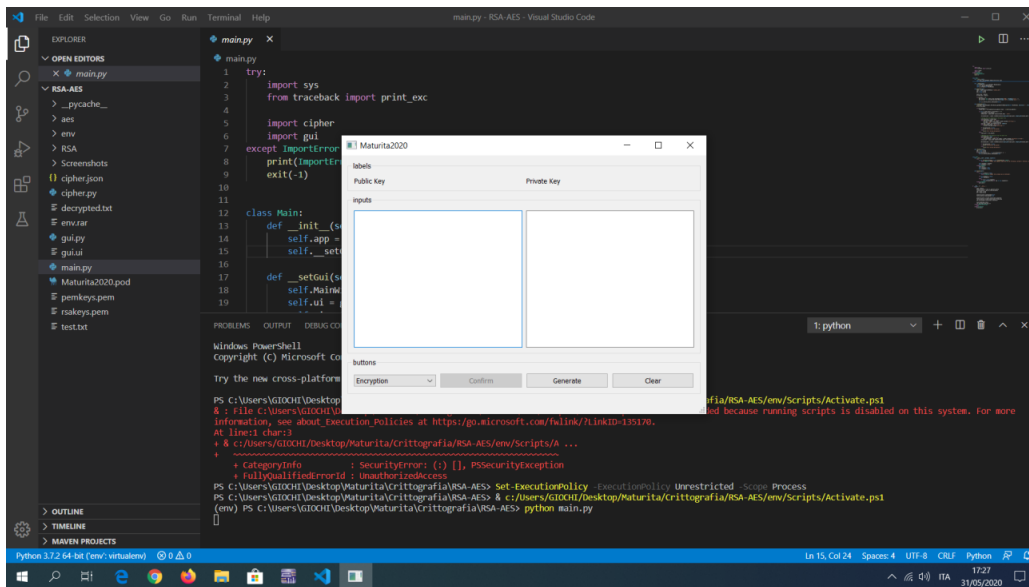
L'utilizzo di container e layout ci aiuta a rendere l'interfaccia resizable in pochi minuti, raggruppando in blocchi i quali, a loro volta, vengono posti sotto la gestione di un layout che ne decide il posizionamento e le dimensioni su tutta l'interfaccia grafica. Il layout che gestisce tutti i container è definito Top-Layout e nel nostro caso si tratta di un layout verticale. Ad ogni container, poi, può essere associato uno specifico layout, nel nostro ogni container è stato posto sotto la gestione, oltre che del Top-Layout, anche di Secondary-Layout, questa volta orizzontale. Per maggiori informazioni sui layout in Qt Designer e sulle loro tipologie, consultare la documentazione al seguente link:

<https://doc.qt.io/qt-5/designer-layouts.html>



Versione finale dell'interfaccia grafica, avviata dopo aver convertito il file .ui generato da Qt Designer al salvataggio in codice Python, utilizzando lo strumento Pyuic5, con il seguente comando:

```
>> pyuic5 -x gui.ui -o gui.py
```



### III - Fase 2

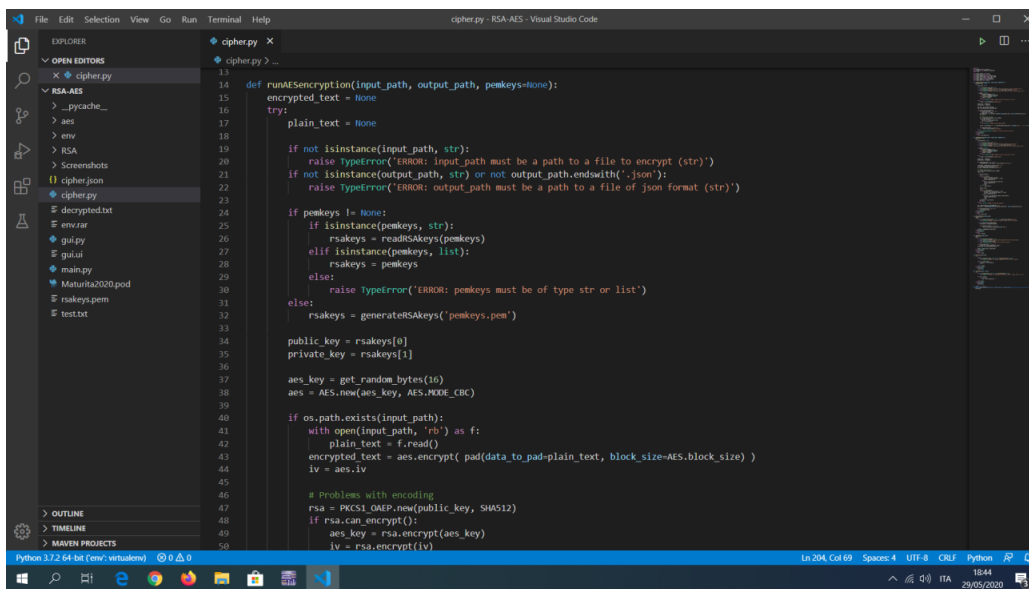
#### Implementazione delle funzioni di cifratura-decifratura

Nell'implementazione algoritmica la scelta del linguaggio è ricaduta ancora una volta su Python, in particolare sull'utilissima libreria PyCrypto, divenuta uno standard di fatto.

In questa fase sono state però prese in considerazione le problematiche che la tipizzazione dinamica di Python e la sua stessa gestione della memoria in runtime, prettamente single-threaded (l'interprete Python non può, per limitazioni imposte dai creatori del linguaggio, interpretare e eseguire più di un thread alla volta, a meno di non utilizzare moduli come multiprocessing. Per ulteriori informazioni: <https://realpython.com/python-gil/>), potrebbero causare. Ciò nonostante, si è arrivati alla conclusione che i vantaggi nell'utilizzare Python sarebbero stati superiori agli svantaggi in questo caso, in quanto la generazione della chiave RSA (da 2048 bit) avviene solo alla prima esecuzione e i file vengono ogni volta criptati con una chiave AES diversa per ogni file. Il payload di ogni file viene quindi cifrato usando un algoritmo simmetrico, molto meno dispendioso in termini di risorse rispetto a RSA che invece viene usato solo per cifrare e decifrare la chiave AES una volta salvata su file.

Per un utilizzo invece su piattaforme hardware molto più limitate dal punto di vista delle risorse, si consiglia una riscrittura delle funzioni in C++ il quale, supportando il paradigma Object Oriented, permetterebbe un più comodo interfacciamento con gli oggetti della libreria PyCrypto, nel caso si decidesse di optare per la scrittura di alcuni bindings tra i due linguaggi (si ricorda che circa la metà dei moduli appartenenti alla Python Standard Library è scritta in C, come anche lo stesso Python Memory Manager: <https://docs.python.org/3/c-api/memory.html>).

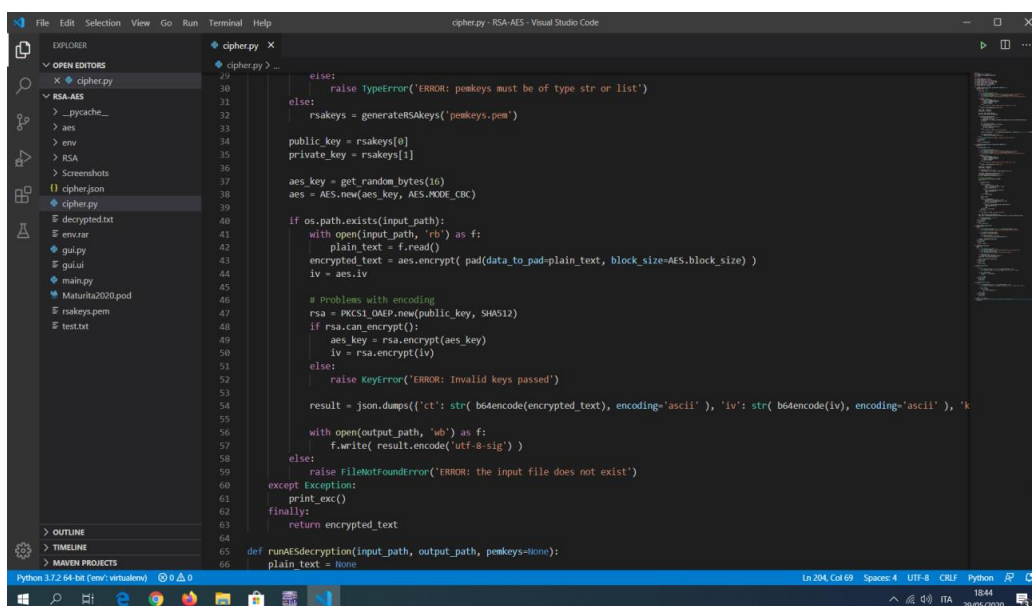
Si procede adesso con l'analisi di alcuni screenshots che mostrano la risoluzione dei problemi più rilevanti occorsi durante questa fase dello sviluppo:



```
13
14
15 def runAESSyncryption(input_path, output_path, pemkeys=None):
16     encrypted_text = None
17     try:
18         plain_text = None
19
20         if not isinstance(input_path, str):
21             raise TypeError('ERROR: input_path must be a path to a file to encrypt (str)')
22         if not isinstance(output_path, str) or not output_path.endswith('.json'):
23             raise TypeError('ERROR: output_path must be a path to a file of json format (str)')
24
25         if pemkeys != None:
26             if isinstance(pemkeys, str):
27                 rsakeys = readRSAKeys(pemkeys)
28             elif isinstance(pemkeys, list):
29                 rsakeys = pemkeys
30             else:
31                 raise TypeError('ERROR: pemkeys must be of type str or list')
32         else:
33             rsakeys = generateRSAKeys('pemkeys.pem')
34
35         public_key = rsakeys[0]
36         private_key = rsakeys[1]
37
38         aes_key = get_random_bytes(16)
39         aes = AES.new(aes_key, AES.MODE_CBC)
40
41         if os.path.exists(input_path):
42             with open(input_path, 'rb') as f:
43                 plain_text = f.read()
44             encrypted_text = aes.encrypt(pad(data_to_pad=plain_text, block_size=AES.block_size))
45             iv = aes.iv
46
47             # Problems with encoding
48             rsa = PKCS1_OAEP.new(public_key, SHA512)
49             if rsa.can_encrypt():
50                 aes_key = rsa.encrypt(aes_key)
51                 iv = rsa.encrypt(iv)
```

Per la generazione della coppia di chiavi RSA si sfrutta il parametro pemkeys. Di default è impostato a None (null), perché non obbligatorio. Di default le chiavi vengono generate sfruttando la funzione generateRSAKeys(output\_path: str) a cui viene passato un parametro, output\_path che indica alla funzione dove salvare le chiavi una volta generate. Il parametro dovrà quindi essere un path di un file in formato pem. Per ulteriori informazioni su questo formato, consultare <https://documentation.progress.com/output/DataDirect/hybridpipeinstall/index.html#page/install/pem-file-format.html>.

Altri parametri passati alla funzione runAESSyncryption sono input\_path, path al file dove prendere i dati in chiaro da criptare, e output\_path, path al file dove salvare i dati cifrati insieme ad altre informazioni come l'IV (Initialitation Vector) e la chiave AES utilizzata, generata attraverso la funzione Crypto.Random.get\_random\_bytes(size: int).



```
29 else:
30     raise TypeError('ERROR: pemkeys must be of type str or list')
31 else:
32     rsakeys = generateRSAkeys('pemkeys.pem')
33
34     public_key = rsakeys[0]
35     private_key = rsakeys[1]
36
37     aes_key = get_random_bytes(16)
38     aes = AES.new(aes_key, AES.MODE_CBC)
39
40     if os.path.exists(input_path):
41         with open(input_path, 'rb') as f:
42             plain_text = f.read()
43             encrypted_text = aes.encrypt( pad(data_to_pad=plain_text, block_size=AES.block_size) )
44             iv = aes.iv
45
46             # Problems with encoding
47             rsa = PKCS1_OAEP.new(public_key, SHA512)
48             if rsa.can_encrypt():
49                 aes_key = rsa.encrypt(aes_key)
50                 iv = rsa.encrypt(iv)
51             else:
52                 raise KeyError('ERROR: Invalid keys passed')
53
54             result = json.dumps({'ct': str( b64encode(encrypted_text, encoding='ascii') ), 'iv': str( b64encode(iv, encoding='ascii') ), 'k'
55
56             with open(output_path, 'wb') as f:
57                 f.write( result.encode('utf-8-sig') )
58         else:
59             raise FileNotFoundError('ERROR: the input file does not exist')
60     except Exception:
61         print_exc()
62     finally:
63         return encrypted_text
64
65 def runAESdecryption(input_path, output_path, pemkeys=None):
66     plain_text = None
```

Come si può notare da una parte di codice commentata, durante la codifica delle stringhe di bytes sono sorti alcuni problemi.

In primo luogo è stato necessario utilizzare le funzioni `Crypto.Util.Padding.pad-unpad(data: bytes, block_size: int)` per poter criptare con AES i dati che, si ricorda, devono necessariamente essere suddivisibili in blocchi da 16 bytes ciascuno (in quanto la chiave AES generata è da 128 bits). La modalità utilizzata per AES, il quale si ricorda essere un cifrario a blocchi, è la Cipher Block Chaining (CBC). Questa modalità prevede che venga generato, al primo blocco di 16 bytes da criptare, un vettore d'inizializzazione (iv, initialisation vector) randomico della stessa dimensione dei blocchi (quindi 16 bytes). Questo per evitare attacchi di frequenza al cifrario, a cui invece è vulnerabile la più semplice modalità ECB (Electronic Code Book). Si effettua poi successivamente l'operazione logica XOR tra i bytes dell'iv e i bytes del primo blocco. Il risultato viene cifrato con la chiave AES. Si ripete poi la stessa identica operazione con tutti gli altri blocchi.

Questa tecnica impedisce al testo, una volta cifrato, di esporre correlazioni tra i blocchi cifrati. Se infatti si utilizzasse sempre la stessa chiave AES per cifrare ogni blocco, accadrebbe che blocchi di bytes che descrivono gli stessi dati produrrebbero lo stesso testo cifrato. Questo ovviamente, è assolutamente da evitare, in quanto esporrebbe il testo ai pericoli di un'analisi crittografica. Se invece si utilizza un blocco di bytes generato randomicamente, l'iv appunto, e si effettua l'operazione di XOR con il primo blocco di bytes del testo in chiaro, cifrando poi il risultato, si ovvia a questa problematica.

L'importanza del vettore d'inizializzazione sta nel fatto che se si effettuasse lo XOR semplicemente tra il primo e il secondo blocco di bytes del testo in chiaro, ci si esporrebbe a un replay attack durante un'ipotetica comunicazione (che nel nostro caso non avviene) tra un mittente, un destinatario e un "terzo incomodo", che potrebbe semplicemente intercettare il messaggio del

mittente e re-inviarlo al destinatario, spacciandosi per il mittente. L'assenza dell'iv potrebbe a quel punto trarre in inganno il destinatario che, trovandosi di fronte allo stesso testo cifrato correttamente (dalla stessa chiave AES), potrebbe rispondere alla richiesta del man in the middle e cadere così nella trappola.

Per ulteriori informazioni su cifrari a blocchi (block ciphers), cifrari a flusso (stream ciphers) e modalità operative, consultare:

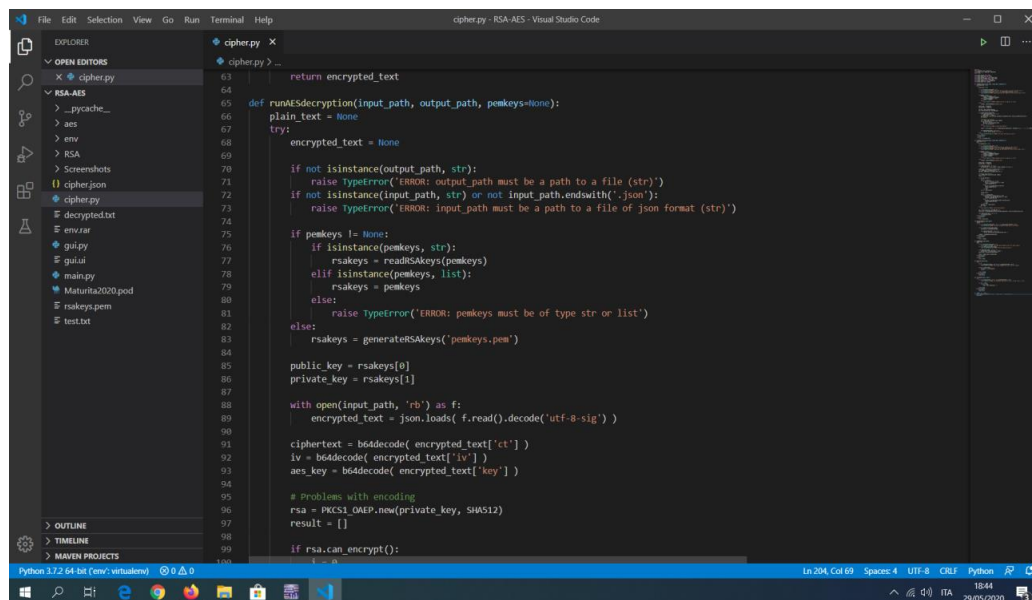
- [https://www.ibm.com/support/knowledgecenter/SSGU8G\\_12.1.0/com.ibm.sec.doc/ids\\_en\\_010.htm](https://www.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.sec.doc/ids_en_010.htm)
- <https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#cbc-mode>

Infine, una volta cifrato il testo, cifrato l'iv e la chiave AES con la chiave pubblica RSA, è stato necessario codificare i bytes nel formato Byte64 e poi effettuare un semplice casting da byte a stringa utilizzando la funzione `str(data: Union[bytes, bytearray, int], encoding: str)`, codificando i dati binari, ovviamente, in ASCII.

Questo per evitare problemi con il formato JSON (i dati vengono salvati come stringa JSON e scritti in un file JSON), il quale non supporta conversioni di dati binari nella funzione `json.dumps(object: object, indent: int)`.

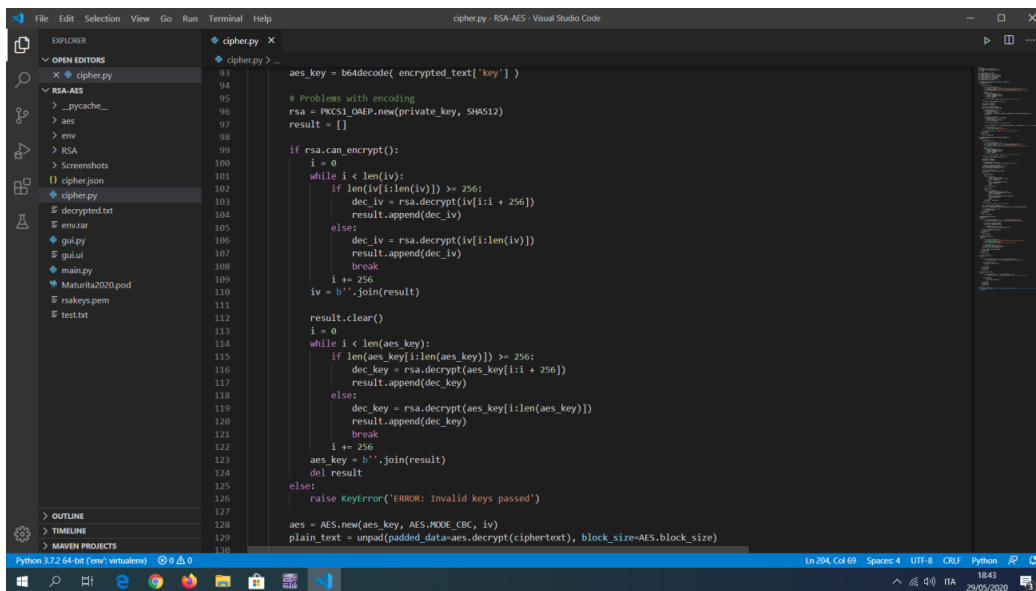
Per ulteriori approfondimenti riguardo la codifica Byte64, ASCII o il formato JSON, consultare:

- <https://it.wikipedia.org/wiki/Base64>
- <https://docs.python.org/3/library/base64.html>
- [https://it.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://it.wikipedia.org/wiki/JavaScript_Object_Notation)
- <https://docs.python.org/3/library/json.html>
- <https://en.wikipedia.org/wiki/ASCII>



```
1  # cipher.py
2
3  def runAESdecryption(input_path, output_path, pemkeys=None):
4      plain_text = None
5      try:
6          encrypted_text = None
7          if not isinstance(output_path, str):
8              raise TypeError('ERROR: output_path must be a path to a file (str)')
9          if not isinstance(input_path, str) or not input_path.endswith('.json'):
10             raise TypeError('ERROR: input_path must be a path to a file of json format (str)')
11
12         if pemkeys != None:
13             if isinstance(pemkeys, str):
14                 rsakeys = readRSakeys(pemkeys)
15             elif isinstance(pemkeys, list):
16                 rsakeys = pemkeys
17             else:
18                 raise TypeError('ERROR: pemkeys must be of type str or list')
19         else:
20             rsakeys = generateRSakeys('rsakeys.pem')
21
22         public_key = rsakeys[0]
23         private_key = rsakeys[1]
24
25         with open(input_path, 'rb') as f:
26             encrypted_text = json.loads(f.read().decode('utf-8-sig'))
27
28         ciphertext = b64decode(encrypted_text['ct'])
29         iv = b64decode(encrypted_text['iv'])
30         aes_key = b64decode(encrypted_text['key'])
31
32         # Problems with encoding
33         rsa = PKCS1_OAEP.new(private_key, SHA512)
34         result = []
35         if rsa.can_encrypt():
36             return encrypted_text
37     except Exception as e:
38         print(e)
```

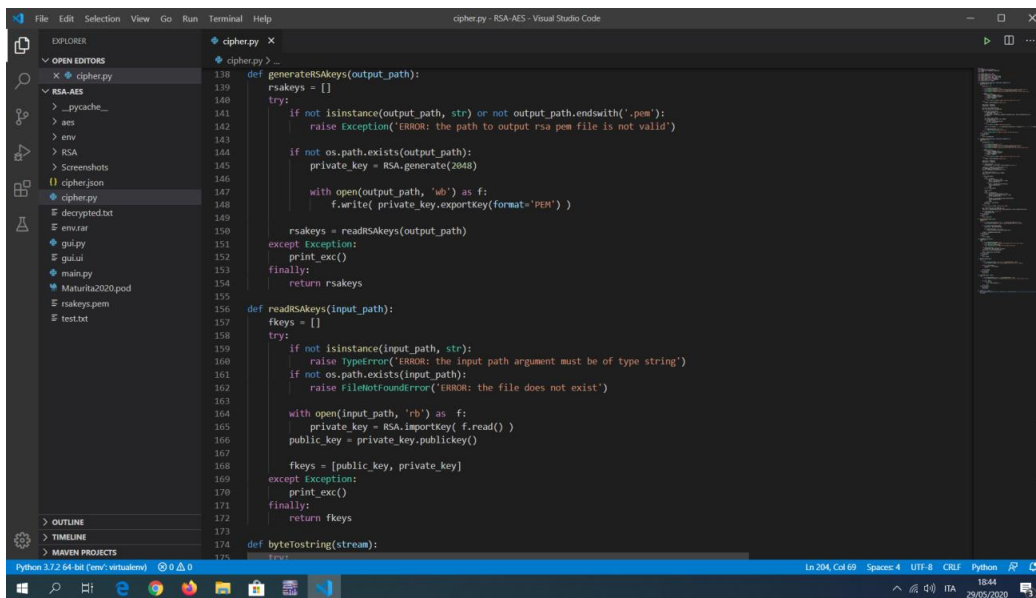




```
93 aes_key = b64decode(encrypted_text['key'])
94
95 # Problem with encoding
96 rsa = PKCS1_OAEP.new(private_key, SHA512)
97 result = []
98
99 if rsa.can_encrypt():
100     i = 0
101     while i < len(iv):
102         if len(iv[i:len(iv)]) >= 256:
103             dec_iv = rsa.decrypt(iv[i:i + 256])
104             result.append(dec_iv)
105         else:
106             dec_iv = rsa.decrypt(iv[i:len(iv)])
107             result.append(dec_iv)
108             break
109         i += 256
110     iv = b''.join(result)
111
112     result.clear()
113     i = 0
114     while i < len(aes_key):
115         if len(aes_key[i:len(aes_key)]) >= 256:
116             dec_key = rsa.decrypt(aes_key[i:i + 256])
117             result.append(dec_key)
118         else:
119             dec_key = rsa.decrypt(aes_key[i:len(aes_key)])
120             result.append(dec_key)
121             break
122         i += 256
123     aes_key = b''.join(result)
124     del result
125 else:
126     raise KeyError('ERROR: Invalid keys passed')
127
128 aes = AES.new(aes_key, AES.MODE_CBC, iv)
129 plain_text = unpad(padded_data=aes.decrypt(ciphertext), block_size=AES.block_size)
130
```

Un aspetto importante della funzione runAESdecryption è questa porzione di codice: come si può notare viene impiegato un ciclo per decifrare con RSA sia l'iv che la chiave AES, decifrando in blocchi da dimensione massima di 256 bytes fino ad arrivare a un blocco di dimensione inferiore.

Questo perché la chiave privata RSA generata precedentemente è da 2048 bits, 256 bytes appunto. RSA non è in grado di cifrare-decifrare blocchi di dati di dimensione superiore a quella della chiave.



```
138 def generateRSAkeys(output_path):
139     rsakeys = []
140     try:
141         if not isinstance(output_path, str) or not output_path.endswith('.pem'):
142             raise Exception('ERROR: the path to output rsa pem file is not valid')
143         if not os.path.exists(output_path):
144             private_key = RSA.generate(2048)
145             with open(output_path, 'w') as f:
146                 f.write(private_key.exportkey(format='PEM'))
147             rsakeys = readRSAkeys(output_path)
148     except Exception:
149         print_exc()
150     finally:
151         return rsakeys
152
153 def readRSAkeys(input_path):
154     fkeys = []
155     try:
156         if not isinstance(input_path, str):
157             raise TypeError('ERROR: the input path argument must be of type string')
158         if not os.path.exists(input_path):
159             raise FileNotFoundError('ERROR: the file does not exist')
160         with open(input_path, 'rb') as f:
161             private_key = RSA.importKey(f.read())
162             public_key = private_key.publickey()
163             fkeys = [public_key, private_key]
164     except Exception:
165         print_exc()
166     finally:
167         return fkeys
168
169 def byteToString(stream):
170     return stream
```

In quest'ultimo screenshots, vediamo la definizione delle funzioni generateRSAkeys e readRSAkeys. Entrambe ritornano una lista contenente al primo elemento l'oggetto di classe `RsaKey:PublicKey`, al secondo l'oggetto di classe `RsaKey:PrivateKey`.

NOTA: In questo caso forse sarebbe stato meglio ritornare una tupla (tuple) e non una lista, in quanto le liste in Python sono mutable objects (oggetti mutabili), mentre le tuple sono immutabili. Le chiavi RSA ritornate sono delle costanti e sarebbe importante evitare una loro involontaria modifica da parte di un programmatore poco competente.



## IV – FASE 3

Scrittura del file `main.py`, gestione degli eventi attraverso le funzioni del modulo (libreria) `cipher.py` ed esecuzione dell'applicativo

Conclusasi la parte di scrittura del modulo `cipher.py`, non resta che utilizzare le funzioni di tale modulo per gestire gli eventi `MouseClicked` sui bottoni dell'interfaccia.

Prima però, è necessario organizzarsi meglio. Si ricorda infatti che il modulo `gui.py` viene rigenerato ogni qual volta si vuole ridisegnare l'interfaccia grafica e tutte le modifiche apportate dall'utente vengono perse ad ogni rigenerazione.

Per evitare quindi di dover ogni volta riscrivere le stesse righe di codice, si consiglia di creare un altro modulo, nel nostro caso direttamente il modulo `main.py` (ma può essere un qualsiasi altro modulo nel caso di applicativi più complessi), ove apportare tutte le modifiche necessarie all'interfaccia grafica.

Per farlo bisogna importare il modulo `gui.py` attraverso l'istruzione `"import"`, l'equivalente Python di `"#include"`, in C/C++. In questo caso non è stato creato alcun file `__init__.py` all'interno della working directory, quindi l'interprete Python non la tratta come un package. Nel caso si volesse però creare una struttura ad albero, con più moduli inseriti in altre sotto-directories, sarebbe a quel punto necessario approfondire i concetti di modulo, package, absolute import, relative import. Per farlo, visionare come sempre la documentazione ufficiale di Python, dove tutto è spiegato nei minimi dettagli:

<https://docs.python.org/3/tutorial/modules.html>

L'interprete Python, di default, quando si richiede l'importazione di un modulo, segue una linea gerarchica durante la ricerca:

- All'interno di `sys.modules`, la cache di tutti i moduli precedentemente importati
- All'interno della Python Standard Library
- All'interno della directory list definita da `sys.path`, la quale contiene anche la current working directory, ove il nostro modulo è situato

Documentazione dal blog `realpython.com`:

<https://realpython.com/absolute-vs-relative-python-imports/#how-imports-work>

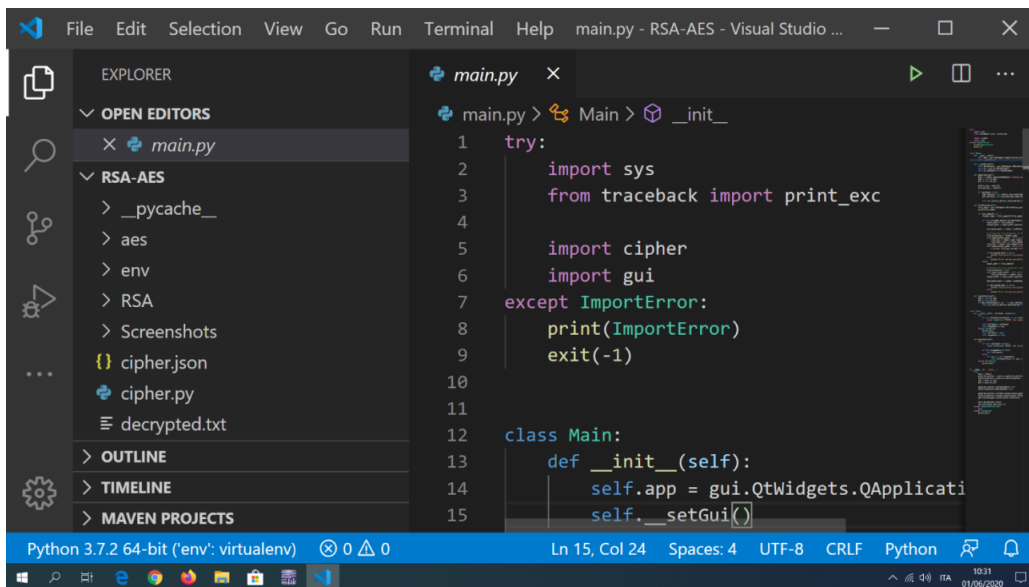
Se il modulo viene localizzato, un puntatore ad esso viene creato e assegnato a una variabile in local scope (disponibile solo all'interno della sessione corrente). Altrimenti viene lanciata un'eccezione, `"ModuleNotFoundError"`.

Documentazione ufficiale di Python sugli import statements:

<https://docs.python.org/3/reference/import.html>

Il concetto di puntatore è molto antico nella programmazione e proviene dal linguaggio C. Il passaggio per riferimento o indirizzo (che fa riferimento ad esso) è presente in tutti i linguaggi, anche in quelli di più alto livello, pertanto è fondamentale capirlo bene:

- <https://www.inf.unibz.it/~calvanese/teaching/01-02-fond-elN/lezioni/3-5-C-puntatori.pdf>
- <https://www.html.it/pag/15412/i-puntatori/>



```
main.py
1  try:
2      import sys
3      from traceback import print_exc
4
5      import cipher
6      import gui
7  except ImportError:
8      print(ImportError)
9      exit(-1)
10
11
12  class Main:
13      def __init__(self):
14          self.app = gui.QtWidgets.QApplicati
15          self.__setGui()
```

Una volta effettuati gli imports, si vuole creare una classe Main, ove inserire tutti i metodi per gestire gli eventi sull'interfaccia grafica.

Si nota che in questo caso non era strettamente necessario creare una classe in quanto si sarebbe potuto tranquillamente definire le funzioni separatamente, all'interno del modulo main.py o in un altro modulo.

Nonostante questo, si consiglia di sfruttare sempre, fin dai primi progetti, il paradigma Object Oriented del linguaggio Python in quanto veramente potente e in grado, se usato al meglio, di semplificare notevolmente la vita nei progetti più complessi.

Nel caso si volesse approfondire ulteriormente la programmazione OO in Python, consultare la documentazione ufficiale:

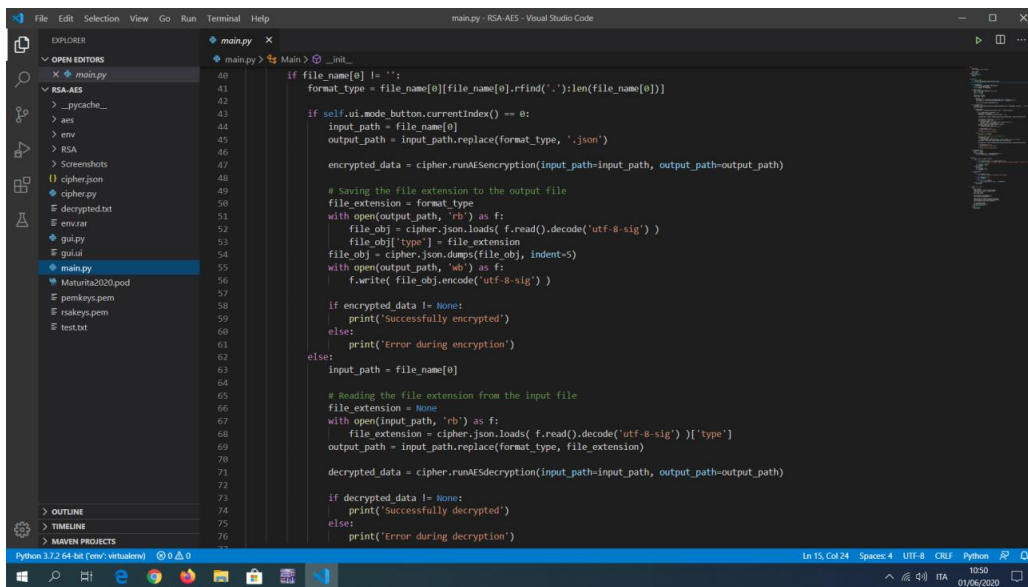
<https://docs.python.org/3/tutorial/classes.html>

```
11 class Main:
12     def __init__(self):
13         self.app = gui.QWidgets.QApplication(sys.argv)
14         self.__setGui()
15         self.__setGui()
16
17     def __setGui(self):
18         self.MainWindow = gui.QWidgets.QMainWindow()
19         self.ui = gui.UI_MainWindow()
20         self.ui.setupUi(self.MainWindow)
21
22     def generate(self):
23         keys = cipher.generateRSAkeys('rsakeys.pem')
24         pbk = self.ui.pbk
25         prk = self.ui.prk
26
27         public_key = keys[0]
28         private_key = keys[1]
29
30         if len(keys) != 0:
31             pbk.setText( str( public_key.exportKey(format='PEM'), encoding='ascii' ) )
32             prk.setText( str( private_key.exportKey(format='PEM'), encoding='ascii' ) )
33             self.ui.confirm_button.setEnabled(True)
34
35     def fileDialog(self):
36         file_name = gui.QWidgets.QFileDialog.getOpenFileName(parent=self.MainWindow, directory='', filter='*.pdf;*.jpg;*.*)')
37         print(file_name)
38
39         if file_name[0] != '':
40             format_type = file_name[0].rfind('.'):len(file_name[0])
41
42             if self.ui.mode_button.currentIndex() == 0:
43                 input_path = file_name[0]
44                 output_path = input_path.replace(format_type, '.json')
45                 encrypted_data = cipher.runAESencryption(input_path=input_path, output_path=output_path)
```

Il primo metodo che si definisce, oltre al costruttore `__init__` e al metodo strongly private `__setGui`, per istanziare e impostare la GUI, è il metodo `generate()`. Con esso si generano le chiavi RSA e le si scrive all'interno delle due text boxes, sbloccando poi successivamente il bottone `confirm_button`, che alla prima esecuzione, è disattivato, come si può notare da quest'altro screenshot:

```
97 if __name__ == '__main__':
98     try:
99         main = Main()
100         generate_button = main.ui.generate_button
101         confirm_button = main.ui.confirm_button
102         pbk = main.ui.pbk
103         prk = main.ui.prk
104
105         generate_button.setEnabled(True)
106         confirm_button.setEnabled(True)
107
108         generate_button.clicked.connect(main.generate)
109         confirm_button.clicked.connect(main.fileDialog)
110         pbk.textChanged.connect(main.checkText)
111
112         main.MainWindow.show()
113         sys.exit(main.app.exec_())
114     except KeyboardInterrupt:
115         pass
116     except Exception:
117         print_exc()
```

Successivamente si definiscono anche i metodi `fileDialog()` e `checkText()`, con i quali, rispettivamente si va ad aprire un file dialog per la scelta di un file qualsiasi da parte dell'utente e si controllano le text boxes, nel caso il loro contenuto sia vuoto (ad esempio, con il bottone "clear" è possibile ripulirle), il bottone "confirm" deve essere nuovamente disattivato.



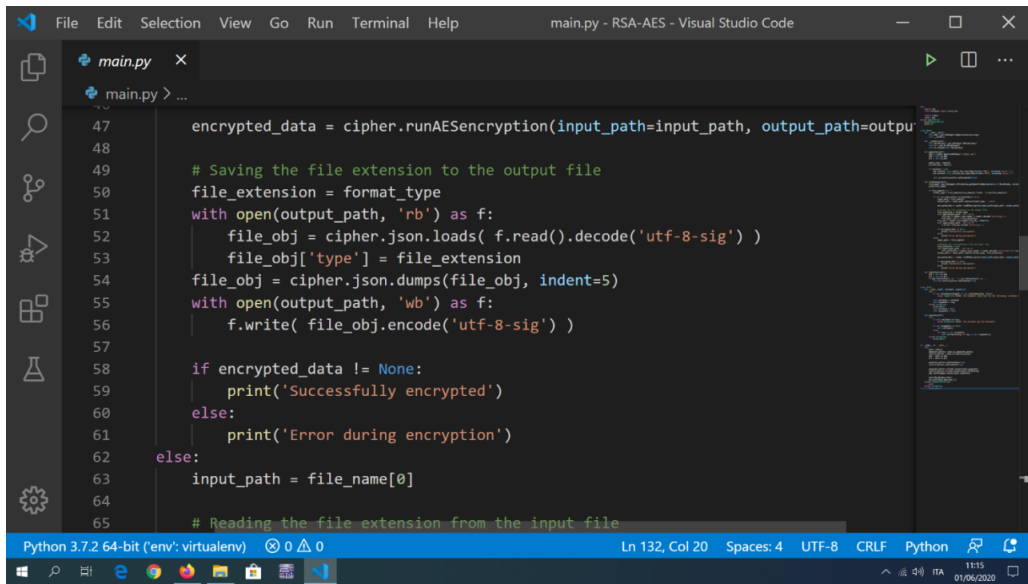
```
40 if file_name[0] != '':
41     format_type = file_name[0][file_name[0].rfind('.'):len(file_name[0])]
42
43     if self.ui.mode.button.currentIndex() == 0:
44         input_path = file_name[0]
45         output_path = input_path.replace(format_type, '.json')
46
47         encrypted_data = cipher.runAESEncryption(input_path=input_path, output_path=output_path)
48
49         # Saving the file extension to the output file
50         file_extension = format_type
51         with open(output_path, 'rb') as f:
52             file_obj = cipher.json.loads( f.read().decode('utf-8-sig') )
53             file_obj['type'] = file_extension
54             file_obj = cipher.json.dumps(file_obj, indent=5)
55             with open(output_path, 'wb') as f:
56                 f.write( file_obj.encode('utf-8-sig') )
57
58         if encrypted_data != None:
59             print('Successfully encrypted')
60         else:
61             print('Error during encryption')
62     else:
63         input_path = file_name[0]
64
65         # Reading the file extension from the input file
66         file_extension = None
67         with open(input_path, 'rb') as f:
68             file_extension = cipher.json.loads( f.read().decode('utf-8-sig') )['type']
69         output_path = input_path.replace(format_type, file_extension)
70
71         decrypted_data = cipher.runAESDecryption(input_path=input_path, output_path=output_path)
72
73         if decrypted_data != None:
74             print('Successfully decrypted')
75         else:
76             print('Error during decryption')
```

Una volta selezionato il file di input con cui operare, all'interno del metodo `fileDialog` vengono richiamate le funzioni del modulo `cipher.py` ed eseguite.

Come argomento al parametro `input_path` viene passato il file selezionato dall'utente, `output_path` sarà invece il file di output che verrà generato all'interno della stessa directory in cui è stato selezionato il file.

Importante è la gestione delle estensioni: sarà necessario infatti salvare l'estensione del file originale nel file di output `.json`, in modo poi da averla disponibile durante la decifatura.

Per farlo si è deciso di operare nel seguente modo:



```
47 encrypted_data = cipher.runAESEncryption(input_path=input_path, output_path=output_path)
48
49 # Saving the file extension to the output file
50 file_extension = format_type
51 with open(output_path, 'rb') as f:
52     file_obj = cipher.json.loads( f.read().decode('utf-8-sig') )
53     file_obj['type'] = file_extension
54     file_obj = cipher.json.dumps(file_obj, indent=5)
55     with open(output_path, 'wb') as f:
56         f.write( file_obj.encode('utf-8-sig') )
57
58 if encrypted_data != None:
59     print('Successfully encrypted')
60 else:
61     print('Error during encryption')
62
63 else:
64     input_path = file_name[0]
65
66 # Reading the file extension from the input file
```

```

63 input_path = file_name[0]
64
65 # Reading the file extension from the input file
66 file_extension = None
67 with open(input_path, 'rb') as f:
68     file_extension = cipher.json.loads( f.read().decode('utf-8-sig') )['type']
69 output_path = input_path.replace(format_type, file_extension)
70
71 decrypted_data = cipher.runAESdecryption(input_path=input_path, output_path=output_path)
72
73 if decrypted_data != None:
74     print('Successfully decrypted')
75 else:
76     print('Error during decryption')
77
78 t(self):
79     lf.ui.pbk
80     lf.ui.prk
81 oPlainText() == '' or prk.toPlainText() == ''

```

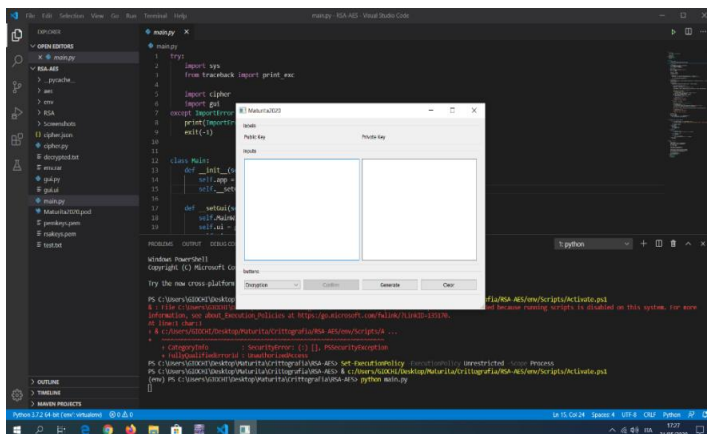
Una volta definito anche il metodo checkText():

```

72
73 if decrypted_data != None:
74     print('Successfully decrypted')
75 else:
76     print('Error during decryption')
77
78 def checkText(self):
79     pbk = self.ui.pbk
80     prk = self.ui.prk
81     if p class Main xt() == '' or prk.toPlainText() == '':
82         self.ui.confirm_button.setDisabled(True)
83
84
85 class Slot:
86     def __init__(self, callback, arg=None):

```

L'applicativo è pronto per essere eseguito.

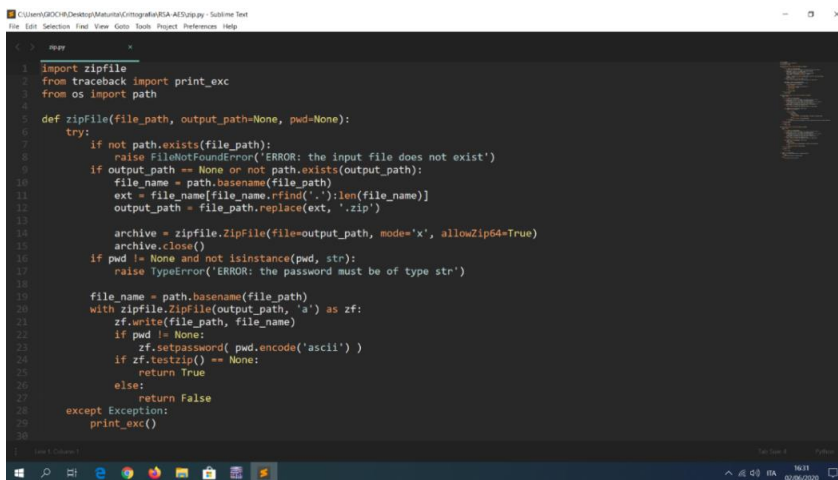


## V – FASE 4

### Implementazione delle funzioni di archiviazione e compressione

Nel caso in cui si preveda di utilizzare l'applicativo per cifrare grandi quantità di dati (indicativamente di dimensione superiore ai 4GB), si suggerisce di implementare delle funzioni di archiviazione e compressione per ridurre lo spazio occupato dai dati una volta cifrati. Per farlo, si può aggiungere un ulteriore modulo, “zip.py” e importare, al suo interno, il modulo della Python Standard Library “zipfile”. Ulteriori informazioni su questo modulo si possono trovare, come per tutti gli altri, nella documentazione ufficiale di Python:

<https://docs.python.org/3/library/zipfile.html>



```
1 import zipfile
2 from traceback import print_exc
3 from os import path
4
5 def zipFile(file_path, output_path=None, pwd=None):
6     try:
7         if not path.exists(file_path):
8             raise FileNotFoundError('ERROR: the input file does not exist')
9         if output_path == None or not path.exists(output_path):
10             file_name = path.basename(file_path)
11             ext = file_name[file_name.rfind('.'):len(file_name)]
12             output_path = file_path.replace(ext, '.zip')
13
14             archive = zipfile.ZipFile(file=output_path, mode='x', allowZip64=True)
15             archive.close()
16             if pwd != None and not isinstance(pwd, str):
17                 raise TypeError('ERROR: the password must be of type str')
18
19             file_name = path.basename(file_path)
20             with zipfile.ZipFile(output_path, 'a') as zf:
21                 zf.write(file_path, file_name)
22                 if pwd != None:
23                     zf.setpassword(pwd.encode('ascii'))
24                 if zf.testzip() == None:
25                     return True
26             else:
27                 return False
28     except Exception:
29         print_exc()
30
```

La prima funzione che è necessario definire è la funzione `zipFile(file_path: str, output_path=None: str, pwd=None: str)`. Con essa viene creato un nuovo archivio, se inesistente, o viene aggiunto un file a un archivio, se quest'ultimo già esiste.

Importante la scelta dei parametri: `file_path` è il file da archiviare e comprimere, `output_path` è il path dove salvare l'archivio .zip. Se di valore `None`, crea l'archivio nella stessa directory di `file_path`. Infine, `pwd` è un eventuale password che se si vuole, si può aggiungere all'archivio. Questa password verrà poi chiesta nuovamente al momento della decompressione.

```
11 def unzipFile(zip_path, name, output_path=None, pwd=None):
12     flag = False
13     try:
14         if not path.exists(zip_path):
15             raise FileNotFoundError('ERROR: the archive does not exist')
16         if output_path == None or not path.exists(output_path):
17             print('WARNING: the output path passed is None or does not exist')
18             print('Setting zip path directory as output path')
19             output_path = zip_path.replace(path.basename(zip_path), '')
20         if pwd != None and not isinstance(pwd, str):
21             raise TypeError('ERROR: the password must be of type str')
22         with zipfile.ZipFile(zip_path, 'r') as zf:
23             files = zf.namelist()
24             for f in files:
25                 if f == name:
26                     flag = True
27                     if pwd != None:
28                         zf.extract(f, path=output_path, pwd=pwd.encode('ascii'))
29                     else:
30                         zf.extract(f, path=output_path)
31             if not flag:
32                 raise FileNotFoundError('ERROR: the file named does not exist in this archive')
33     except Exception:
34         print_exc()
35     finally:
36         return flag
37
38 def unzipall(zip_path, output_path=None, pwd=None):
```

Conseguentemente alla scrittura della prima funzione, ne viene scritta un'altra che consenta di estrarre da un archivio un determinato file:

UnzipFile(zip\_path: str, name: str, output\_path=None: str, pwd=None: str)

I parametri sono gli stessi di prima, eccetto che per name, che specifica il nome del file da ricercare ed eventualmente estrarre (se viene trovato) nell'archivio

```
28     return flag
29
30 def unzipall(zip_path, output_path=None, pwd=None):
31     flag = False
32     try:
33         if not path.exists(zip_path):
34             raise FileNotFoundError('ERROR: the archive does not exist')
35         if output_path == None or not path.exists(output_path):
36             print('WARNING: the output path passed is None or does not exist')
37             print('Setting zip path directory as output path')
38             output_path = zip_path.replace(path.basename(zip_path), '')
39         if pwd != None and not isinstance(pwd, str):
40             raise TypeError('ERROR: the password must be of type str')
41         with zipfile.ZipFile(zip_path, 'r') as zf:
42             flag = True
43             if pwd != None:
44                 zf.extractall(path=output_path, pwd=pwd.encode('ascii'))
45             else:
46                 zf.extractall(path=output_path)
47     except Exception:
48         print_exc()
49     finally:
50         return flag
51
52 if __name__ == '__main__':
53     flag = unzipFile('tandem.1.zip', 'tandem.1.json')
54     if flag:
55         print('Successfully uncompressed')
56     else:
```

Infine, viene creata anche una funzione per estrarre tutti i files da un archivio, a prescindere dal loro nome:

UnzipAll(zip\_path: str, output\_path=None: str, pwd=None: str)