**Seminar 2 Preparation (and e-portfolio entry)**

Read the Cryptography with Python blog at tutorialspoint.com (link is in the reading list). Select one of the methods described/ examples given and create a python program that can take a short piece of text and encrypt it.

Create a python program in Codio (you can use the Jupyter Notebooks space provided in the Codio resources section) that can take a text file and output an encrypted version as a file in your folder on the Codio system. Demonstrate your program operation in this week’s seminar session.

Answer the following questions in your e-portfolio:

* Why did you select the algorithm you chose?
* Would it meet the GDPR regulations? Justify your answer.

We will review your work from Unit 4 in this week’s seminar, as well as this cryptography activity. There will also be an opportunity to review your team’s assignment progress during the seminar.

**Why did you select the algorithm you chose?**

"I choose RSA as this is the strongest on all of them, RSA encryption uses both public and private keys. The opposite key decrypts and communication this is why RSA is most used asymmetric algorithm, it ensures electronic communications secrecy, integrity, authenticity, and non- reputability.

We deploy Encryption to implement confidentiality. GDPR requires EU citizens PII data at all cost. It has about 118 articles which direct Data collectors and processors to adhere in order to protect data... Read the text article and see if there are any EU personnel involved in the case study. If so select their PII data part and encrypt...

**Would it meet the GDPR regulations? Justify your answer:**

"Yes, as under the GDPR, encryption is NOT mandatory, but it is mentioned as one of the mechanisms to protect personal data. RSA being the strongest will help protect sensitive data"

RSA Cipher Encryption

In this chapter, we will focus on different implementation of RSA cipher encryption and the functions involved for the same. You can refer or include this python file for implementing RSA cipher algorithm implementation.

The modules included for the encryption algorithm are as follows −

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA512, SHA384, SHA256, SHA, MD5

from Crypto import Random

from base64 import b64encode, b64decode

hash = "SHA-256"

We have initialized the hash value as SHA-256 for better security purpose. We will use a function to generate new keys or a pair of public and private key using the following code.

def newkeys(keysize):

random\_generator = Random.new().read

key = RSA.generate(keysize, random\_generator)

private, public = key, key.publickey()

return public, private

def importKey(externKey):

return RSA.importKey(externKey)

For encryption, the following function is used which follows the RSA algorithm −

def encrypt(message, pub\_key):

cipher = PKCS1\_OAEP.new(pub\_key)

return cipher.encrypt(message)

Two parameters are mandatory: **message** and **pub\_key** which refers to Public key. A public key is used for encryption and private key is used for decryption.

The complete program for encryption procedure is mentioned below −

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA512, SHA384, SHA256, SHA, MD5

from Crypto import Random

from base64 import b64encode, b64decode

hash = "SHA-256"

def newkeys(keysize):

random\_generator = Random.new().read

key = RSA.generate(keysize, random\_generator)

private, public = key, key.publickey()

return public, private

def importKey(externKey):

return RSA.importKey(externKey)

def getpublickey(priv\_key):

return priv\_key.publickey()

def encrypt(message, pub\_key):

cipher = PKCS1\_OAEP.new(pub\_key)

return cipher.encrypt(message)

RSA Cipher Decryption

This chapter is a continuation of the previous chapter where we followed step wise implementation of encryption using RSA algorithm and discusses in detail about it.

The function used to decrypt cipher text is as follows −

def decrypt(ciphertext, priv\_key):

cipher = PKCS1\_OAEP.new(priv\_key)

return cipher.decrypt(ciphertext)

For public key cryptography or asymmetric key cryptography, it is important to maintain two important features namely **Authentication** and **Authorization**.

Authorization

Authorization is the process to confirm that the sender is the only one who have transmitted the message. The following code explains this −

def sign(message, priv\_key, hashAlg="SHA-256"):

global hash

hash = hashAlg

signer = PKCS1\_v1\_5.new(priv\_key)

if (hash == "SHA-512"):

digest = SHA512.new()

elif (hash == "SHA-384"):

digest = SHA384.new()

elif (hash == "SHA-256"):

digest = SHA256.new()

elif (hash == "SHA-1"):

digest = SHA.new()

else:

digest = MD5.new()

digest.update(message)

return signer.sign(digest)

Authentication

Authentication is possible by verification method which is explained as below −

def verify(message, signature, pub\_key):

signer = PKCS1\_v1\_5.new(pub\_key)

if (hash == "SHA-512"):

digest = SHA512.new()

elif (hash == "SHA-384"):

digest = SHA384.new()

elif (hash == "SHA-256"):

digest = SHA256.new()

elif (hash == "SHA-1"):

digest = SHA.new()

else:

digest = MD5.new()

digest.update(message)

return signer.verify(digest, signature)

The digital signature is verified along with the details of sender and recipient. This adds more weight age for security purposes.

**Reference**

https://www.tutorialspoint.com/cryptography\_with\_python/cryptography\_with\_python\_quick\_guide.htm# (Access 25 of Jun, 2022)

https://www.venafi.com/blog/how-diffie-hellman-key-exchange-different-rsa (Access 25 of Jun ,2022)