Seminar 4

Cryptography Programming Exercise

Q1: Why did you choose this algorithm?

```
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
import os

hash = 'SHA-256'
key_size = 1024

private_key, get_public_key = b"",b""


def new_keys(key_size):
    random = Random.new().read
    key = RSA.generate(key_size, random)

    private, public = key, key.publickey()
    return private, public


def import_key(key):
    return RSA.importKey(key)


def get_public_key(private_key):
    return private_key.publickey()


def encrypt_file(source, destination):
    cipher = PKCS1_OAEP.new(public_key)

    with open(source, "rb") as src, open(destination, "wb") as dest:
        for chunk in iter(lambda: src.read(80), b""):
            encrypted = cipher.encrypt(chunk)
            dest.write(encrypted)
```

```
def decrypt_file(source, destination):
    cipher = PKCS1_OAEP.new(private_key)

    with open(source, "rb") as src, open(destination, "wt") as dest:
        for chunk in iter(lambda: src.read(128), b""):
            decrypted = cipher.decrypt(chunk).decode('utf-8')
            dest.write(decrypted)


def read_content_as_text(source):
    with open(source, "r") as src:
        print(src.read())


def read_content_as_binary(source):
    with open(source, "rb") as src:
        data = src.read().hex()

        print(data)


private_key, public_key = new_keys(key_size)

if __name__ == "__main__":
    filename = "RSA_SampleText.txt"

    print (f"* Before encryption, content is:\n----------------------
----------------------")
    read_content_as_text(filename)

    # Run the encryption example
    destination = os.path.basename(filename)
    name = destination.split('.', 1)[0]
    extension = destination.split('.', 1)[1]
    destination = name + ".encrypt." + extension
    print (f"\n* Encrypting file to: {destination}...\n------------
------------------------------")

    encrypt_file(filename, destination)

    print (f"\n* After encryption, content is:\n--------------------
----------------------")
    read_content_as_binary(destination)

    # Run the decryption example
```

```
    destination2 = name + ".descrypt." + extension

    decrypt_file(destination, destination2)

    print (f"\n* After decryption, content is:\n----------------------
----------------------")
    read_content_as_text(destination2)
```

Program Output

```
* Before encryption, content is:

-----------------------------------------

“RSA (Rivestâ€“Shamirâ€“Adleman) is an algorithm used by modern computers
to encrypt and decrypt messages. It is an asymmetric cryptographic
algorithm. Asymmetric means that there are two different keys. This is also
called public key cryptography, because one of the keys can be given to
anyone. The other key must be kept private. The algorithm is based on the
fact that finding the factors of a large composite number is difficult:
when the factors are prime numbers, the problem is called prime
factorization. It is also a key pair (public and private key) generator.
RSA involves a public key and private key. The public key can be known to
everyone- it is used to encrypt messages. Messages encrypted using the
public key can

only be decrypted with the private key. The private key needs to be kept
secret. Calculating the private key from the public key is very difficult.”


Source: https://simple.wikipedia.org/wiki/RSA_algorithm


* Encrypting file to: RSA_SampleText.encrypt.txt...

-----------------------------------------


* After encryption, content is:

-----------------------------------------

a59c5c5b7de123e091d1e6c1d2445d0e41763e4c1484604f392c813f98cfd4971cc88b17f
7038c1bd313e861066ba172c6363c0693dc970163d66f7f4d27a0ec32c7899341090bf2c7
63e38aee57be5eb7f15fe10ff69b243cfcdbdcfd7de4a695293f6ebabff95f6e0f27b72f8
15bb28c15e1a4ea427fd253fec78a6ab67afdaca77cf777a6bc7db9d69b2c3d8d8fc803b9
ca4d594d0b3cf2bd8bcb682d2abf4744512240807fa2a3ac82dd6ff044cf20a033f7eae16
54b552443c5159b85b0830b7d0baf5fc08bde60b41812a5ef264b42420d3a03cae8fb4f46
efa42db3f222a0585985d807fb0b738ba1b1f11f235a425cd4e3439ebbd70415382a3edc9
```

```
2674290e5fc357d0fd19f6fb5c64e28946e16ecbdb27b579dbb529686f5e813890cab7bce
c4bbed25621d953d5e3df428d95cdc1ee9a925051221dd20843374cfebc246449e17e50e4
f187a3b38e88c35ef0f62dd3203a5e1fb7994be99330de72b5f16c5afafc6c74e7a4bdbcc
c20111b9ab5afad45f14f2dd37cc00b01296cb539b4a78b80ed88cd73bea5096841ffcc90
5e4d188022221e4279fe9096ebd23b89022e188c106d46f33313c514f7a488b1eb4cd1279
f6f35270adb57cf043e57f3010a0ccfd93729065cb7a498f0a20aa08abbde4cd4ff8511d6
157b472d70c34f160379255a71fff34f2700906204a4c88cbcdef1ffc35acad5d508fe293
c55f34c11e2ecd0a2f81a081dd074e807e334d93274d1485746998e1b01ce2485e97fbab5
48c6b5611f57c512eb99d720e244acb4ede95a933a9578df946ff9f12b6c88e0f4183a8f7
6e8b86c969f170b2f3ea3d45380e935f4afb44576861650b3aa9d6827048e2b551bc52a58
d2f217dde552acd120b4634e69136417109e6271c57aecc677506d631a3e47fcd9fbd0d47
acef7413ea7214a78da76a0432357f796f51c53bf08948ebdc43716f54b6f8417ac214eca
d864e1b4519b2258a0eddf016f73480bfd2721012b3e65fc2cd8c5b249420fe1dd2cc0457
cf3e2a98c7a5df8f7c224caca0bd0f2f748d3e3d60335934b9a662310c58a165fa2704ac9
95248c28cbd65951c5a89ae0164068c6d35c3cdd868736666322135c8b864bf3d62188f90
94fca7f0274f57cef832b3ff85045ec96be2a127850ec7fc2a9f6524d59e87dac867b504b
3f9cb79f2e1614f06e48b97177c84eeddd9c36418f02ba1334972c47071350106fbc4a9b5
7a54e2b09d60033afe5b04f27e448ae81c674c200ab4e96924bdba1d73615d8067ead3056
6a132a227ea4ff4dc2664a44320f6d5b31153e3819ed7c4058e330a7e8392a762175b8200
016745e98a11b901a50a100bc1079339b256339b91de34e655fb58251612656510f4bda15
6d001a6b4f1fd7ef8dfa8e39d07bdb9cdf2cc4e4dd478d1ce8a308d9b43c583c027f7b783
38415131f018188aff459a88311cdf91dc0f3dbc524df863f9975266e3559bfb5f8eb4b7a
c593e9560bf5b67c7292235ddd8ffb18e64f8645bf79353d90a402215d21b0fd29d8c1c78
f8503c96319dc5ff80f873509e7e5eeb2095c841bcf30b4c33ebc3a7ed894dee5cf431f70
9922120a5f4ad3d58c81246ad2054e9ae89a96ab82e1b7f08cd34f40857bc740d8f51a470
ba164800db188fb8d1bfbce2e4a9f04abeaac7672625c22f7dfb2a5d31f002b8c6bac005c
9cf5ad46b16ad2f464de78749ac847f0ca6ce340e235bb91327c291dfac3c6e090fc76382
336ff9e3fc309190f2f6a40e963ca2b1f6dfa1d81e18d91905dcdf67f1b6e86d00342b4c9
ceb255a613e56c4b69b64bbff7d6fcc5d2119fdc411ecb51bf725e08cde464bf9c0b1f2af
89a1c26f79bbe9a0514c20e1eefc40953eed775d3bd435c5c278cc6560f3162beeb29bb3c
f91e503cc5681521259b21a2559a88342c90b258c96554461c8b2a86a7b33e9818f7f6947
ec095d6092a3b3cf8a0104a47dbe2782c1ef21f4734be77fb22dd7fe4fe4027d859bc9f32
636c9a4fc06a24b3391374375fcceb6f4a088a92200348dd72ca44ca131dae658420fab20
d6aaf6fbc511bc678c7b52f5177bd342b21403197f0b5e94a8c09e7d199a94794c8e6cc83
1c449647672904d15228be32b838e9f297b5886c7a336963a517a4f8060f2814b51f7dd8f
88ef30


* After decryption, content is:

----------------------------------------

“RSA (Rivest–Shamir–Adleman) is an algorithm used by modern computers to
encrypt and decrypt messages. It is an asymmetric cryptographic algorithm.
Asymmetric means that there are two different keys. This is also called
public key cryptography, because one of the keys can be given to anyone.
The other key must be kept private. The algorithm is based on the fact
that finding the factors of a large composite number is difficult: when
the factors are prime numbers, the problem is called prime factorization.
It is also a key pair (public and private key) generator. RSA involves a
public key and private key. The public key can be
```

```
known to everyone- it is used to encrypt messages. Messages encrypted using
the public key can only be decrypted with the private key. The private key
needs to be kept secret. Calculating the private key from the public key
is very difficult."
```

```
Source: https://simple.wikipedia.org/wiki/RSA_algorithm
```

I chose the RSA algorithm for file encryption, with a key size of 1024 bits. Several easily searchable articles on the internet refer users to use the Fernet algorithm (PyShark, n.d) (CodersHubb, n.d). The appeal for the Fernet algorithm is that it is easy to implement and use (Buchanan, 2018), which uses 128-bit AES in CBC mode and PKCS7 padding. Many of today's CPUs (Intel) have in-built support for AES (Intel, 2012). Even Microsoft's latest update supports AES encryption to protect their MS-SAMR protocol (Microsoft, 2021).

However, a major *downside* to using the Fernet algorithm is that the data to encrypt must fit inside computer memory. Fernet is therefore unsuitable for files that may exceed a few hundred megabytes in size. RSA, in comparison, can handle almost limitless blocks of data, albeit it at the cost of performance. `encrypt_file` and `decrypt_file` demonstrate how to leverage the RSA encryption and decryption routines on potentially large files. Another factor in selecting RSA is use of **public key** and **private key** pairs, thereby ensuring an even higher level of security of the encrypted data. However, the downside is that encryption-key management becomes more critical than the simpler Fernet algorithm.

Q2: Does the algorithm meet GDPR?

While GDPR covers a wide range of requirements related to the processing and storage of data, the selected algorithm most can be leveraged in the *anonymisation* of personal data and can therefore be safely employed in data protection policies. According to Hambley (2019), Article 33 does not apply if encrypted data is lost and the keys to the encrypted data have been breached. In such a scenario, the breach is treated as an unencrypted data breach. He also makes an argument that destruction of keys related to encrypted data is the same as destroying the data itself. Therefore, leveraging a strong encryption algorithm such as RSA is very suitable for use with GDPR protection policies.

Article 32(1) of the General Data Protection Regulations places the responsibility for protecting personal data on the data controller but does not define what steps must be takes to ensure suitable technical measures are adopted to protection of said data. Coos (2021) points out that GDPR does not mandate for use of encryption, but that it is referenced at least four times within the regulation as a recommendation and as an effective technical measure.

References

Buchanan, B. (2018) If you're struggling picking a Crypto suite … Fernet may be the answer. Available from https://medium.com/coinmonks/if-youre-struggling-picking-a-crypto-suite-fernet-may-be-the-answer-95196c0fec4b [Accessed 19 Oct. 2021]

Coos, A. (2021) GDPR Data Encryption Requirements. Available from https://www.endpointprotector.com/blog/gdpr-data-encryption-requirements/ [Accessed 19 Oct. 2021]

CodersHubb (n.d) Encrypt or decrypt files using python. Available from https://www.codershubb.com/encrypt-or-decrypt-files-using-python/ [Accessed 19 Oct. 2021]

Hambley, L. (2019) Using cryptography to protect PII in GDPR protected jurisdictions. Available from https://dev.to/leehambley/using-cryptography-to-protect-pii-in-gdpr-protected-jurisdictions-29kc. [Accessed 19 Oct. 2021]

Intel (2012) Intel Advanced Encryption Standard (Intel® AES) Instructions Set – Rev 3.01. Available from https://www.intel.com/content/www/us/en/developer/articles/tool/intel-advanced-encryption-standard-aes-instructions-set.html [Accessed 19 Oct. 2021]

Microsoft (2021) KB5004605: Update adds AES encryption protections to the MS-SAMR protocol for CVE-2021-33757. Available from https://support.microsoft.com/en-us/topic/kb5004605-update-adds-aes-encryption-protections-to-the-ms-samr-protocol-for-cve-2021-33757-e4daa133-54aa-4a5d-a921-04bb50868fc2 [Accessed 19 Oct. 2021]

PyShark (n.d) Encrypt and Decrypt Files using Python. Available from https://pyshark.com/encrypt-and-decrypt-files-using-python/ [Accessed 19 Oct. 2021]