Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Liyanage Don	Student ID:	21215607
Other name(s):	Nesith Hesara Nethmitha Liyanage		
Unit name:	Fundamental Concepts of Cryptography	Unit ID:	ISEC2000
Lecturer / unit coordinator:	Mr. Roshan	Tutor:	
Date of submission:	18/05/2024	Which assignment?	2 (Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: [illegible] Date of signature: 18/05/2024

(By submitting this form, you indicate that you agree with all the above text.)

Fundamental Concepts of Cryptography – ISEC2000

2024 Semester 1

Assignment 2

Report

Name: Nesith Liyanage

Curtin ID: 21215607

Conceptual Understanding and Question Answering

1. (10 points) Explain the principle of public key cryptography and how it differs from symmetric key cryptography. Also, describe the process of encryption in the RSA algorithm.

Public key cryptography or asymmetric key cryptography is a system that uses pairs of keys, a public key and a private key. Public key is a key that is shared openly, and it is mainly used for encryption. Private key is kept safe and is mainly used for decryption. But symmetric key cryptography is a system that uses one key for both encryption and decryption. The key should be shared only among both the encrypting and decrypting parties and it can be challenging to do it securely.

RSA Algorithm is one of the most widely used public key encryption algorithms.

- First RSA does key generation, by taking the product (n) of two large prime numbers and computing them using Euler's totient function by deriving the encryption and decryption exponents, e and d, such that e.d ≡ 1 mod ϕ (n), ϕ (n) is the Euler's totient function of n.
- Next encryption, the sender accesses the receiver's public key and computes the ciphertext.
- Finally in decryption, the receiver uses his private key to decrypt the ciphertext.

2. (10 points) Given two prime numbers p = 61 and q = 53, calculate the modulus n, Euler's totient function ϕ(n), and select an appropriate public key e, such that (60 ≤ e ≤ 70). Now, proceed to compute the private key d, assuming e = 17.

Take the product of two large prime numbers:

n = p.q = 61*53 = 3233

Compute ϕ (n):

ϕ (n)= (p-1) * (q-1) = (61-1) * (53-1) =3120

find public key e (60 ≤ e ≤ 70):

gcd (e, ϕ (n)) =1

e =61 – gcd (61,3120) =1

therefore, public key e = 61

private key calculation d:

e.d ≡ 1 mod ϕ (n)

d ≡ 1 mod ϕ (n)/e

d ≡ 1 mod 3120/61

d ≡ 1381

3120 =61×51+9
61 =9×6+7
9 =7×1+2
7 =2×3+1

GCD=1

If public key e=17

Private key calculation d:

e.d ≡ 1 mod ϕ (n)

d ≡ 1 mod ϕ (n)/e

d ≡ 1 mod 3120/17

d ≡ 2753

3. (10 points) The Euclidean algorithm is based on the following assertion. Given two integers a, b, (a > b),

gcd(a, b) = gcd(b, a mod b). (1)

Prove the assertion (1) mathematically. (Note that proof by example is NOT appropriate here)

In order to prove the assertion mathematically, Let's take (gcd(a, b)) as (d). By definition, (d) is the greatest common divisor of (a) and (b). Showing that:

1. (d) divides both (a) and (b).

2. Any common divisor of (a) and (b) must divide (d).

Now we should prove that (gcd(b, a mod b)) is also (d).

It can be seen that (a) can be written as (a = bq + r). where (q) is the quotient and (r) is the remainder when (a) is divided by (b).

Since (d) is the greatest common divisor of (a) and (b), it divides both (a) and (b), because (r = a – bq) it must also divide (r) .Therefore, (d) is a common divisor of (b) and (r).

Since (r = a - bq), it's can be seen that any common divisor of (b) and (r) must also divide (a). Showing it divides both (a) and (b), making it a divisor of (gcd (a, b) = d). Therefore, (d') must be less than or equal to (d).

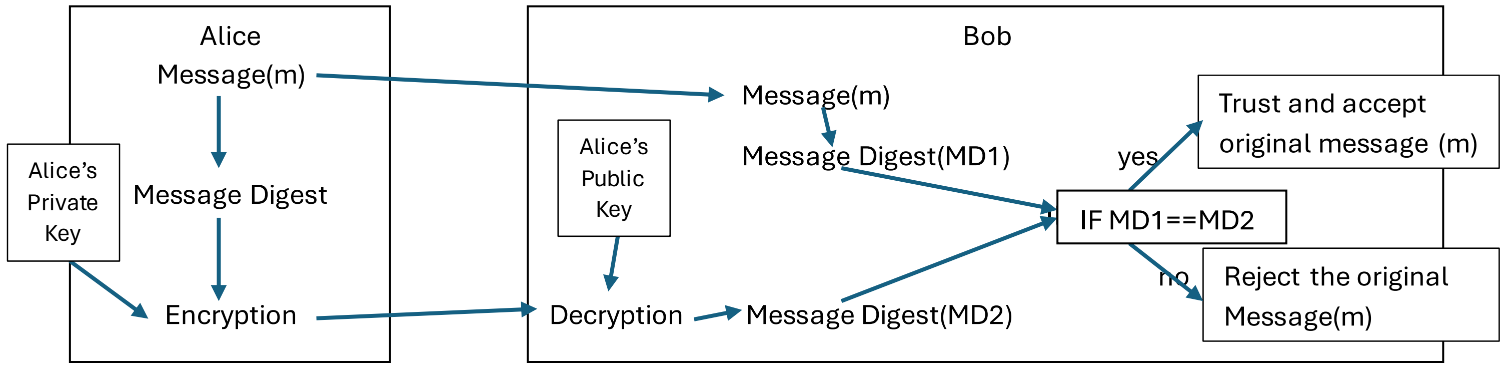
This shows that any common divisor of (b) and (r) must divide (d), and (d) is the greatest common divisor of (b) and (r). Hence, (gcd(b, a mod b) = d). Showing that:

(gcd(a, b) = d), and (gcd(b, a mod b) = d).

Therefore, it shows the Euclidean algorithm is based on the following assertion by proving mathematically that:

(gcd(a, b) = gcd(b, a mod b)).

4. (10 points) Provide a diagram of the RSA signature structure and explain how it is used to authenticate a message. Discuss the steps involved when Alice signs a document m using her private key and how this signature is verified by Bob using Alice's public key.



Alice signs a document by encrypting its message digest (or hash) with her private key, and Bob verifies the signature by decrypting it with Alice's public key and comparing the message digest that is received with the calculated message digest. This process helps to identify the authenticity of the document and the identity of the signer.

Alice signs a document m:
- Alice will first get the message digest or the hash of document m.
- Alice will encrypt the message digest with her private key and create a signature.
- Then pass it with the document showing she signed it.

Verification of signature by Bob:
- Bob gets the Alice's public key from Alice after receiving the document m and the signature.
- Bob decrypts the message digest or the hash of the received document.
- Bob compares the received files digest with the calculated message digest and if:
    - If they match it can be verified the signature is from Alice and the document is maintained with integrity
    - If they DO NOT match the signature is invalid and the document has been tampered with and cannot be trusted.

5. (15 points) Assume that Alice has signed a document m using the RSA signature scheme, and the signature, along with the message, is sent to Bob. Describe a scenario where Bob discovers another message m' (where m ≠ m') such that H(m) = H(m'), with H() being the hash function used in the signature scheme.

- **Describe, in detail, the steps Bob would take to forge Alice's signature for m' using the signature of m.**

Bob can first verify the valid signature using Alice's public key for m
Bob can then use that signature for m' because H(m) = H(m'). It also results in the signature of m' to be equal to the signature of m.

- **Discuss the cryptographic vulnerabilities exploited in this process and the potential impact on the authenticity of signed messages.**

In this situation the weakness of the hash functions collision resistance is shown exposing the cryptographic vulnerabilities exploited. Hash functions are designed to make finding two different inputs with the same hash value computationally infeasible. Which in this situation finding a collision allows an attacker to forge a signature for a different message.

The forged signature for the new message m' can affect the authenticity and integrity of the digital signature scheme. It allows an attacker to impersonate the owner of the signature and sign messages without possessing owner's private key and without the owner's consent. Which can help in unauthorized access, change in data, or any other unauthorized activities under the guise of owner's identity.

Programming

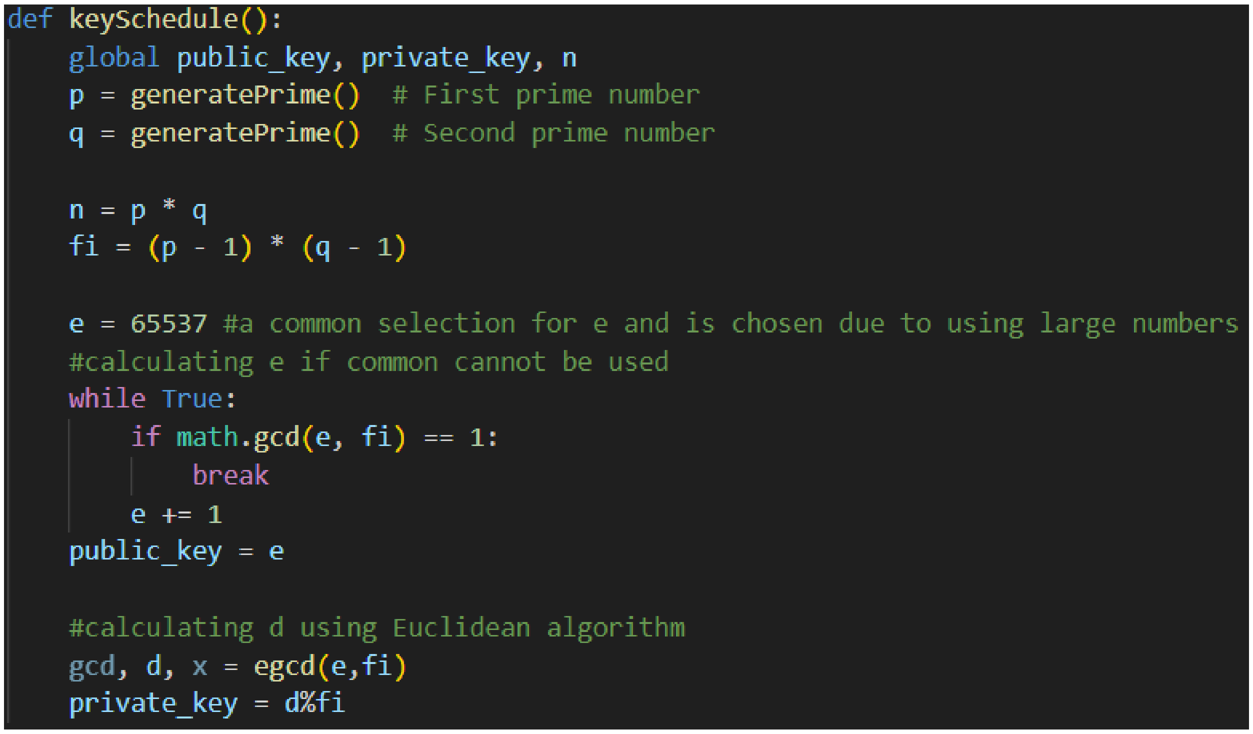
The file named “RSA.py” contains the RSA algorithm program in python, to encrypt and decrypt files using RSA.

The program is implemented with the features:

- Implemented each component as a separate function, such as key schedule, prime test, the extended Euclidean algorithm, binary modular exponentiation, and so on.
- Implemented both encryption and decryption of RSA. Encryption takes a txt file as input and output another txt file containing ciphertext that is converted to hexadecimal for easy readability. And decryption recovers the plaintext.
- The code encrypts and decrypts standard keyboard characters, including letters, numbers, and symbols.
- The prime numbers p and q are larger than 264.

Few functions within the program:

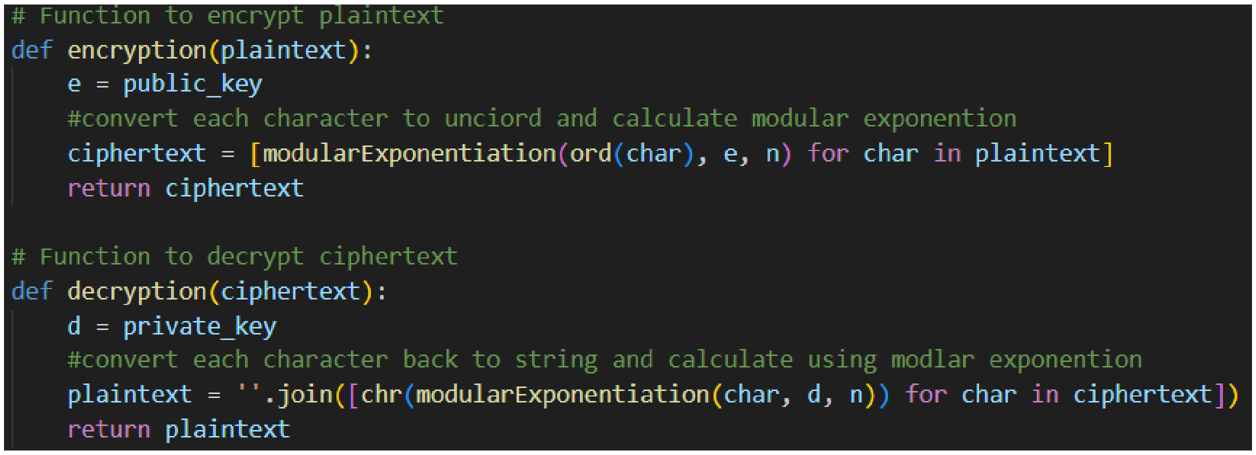
The keySchedule() function is used to generate the public key and private key. Using the GCD and Euclidean theorem to calculate them.



```python
def keySchedule():
    global public_key, private_key, n
    p = generatePrime()  # First prime number
    q = generatePrime()  # Second prime number

    n = p * q
    fi = (p - 1) * (q - 1)

    e = 65537 #a common selection for e and is chosen due to using large numbers
    #calculating e if common cannot be used
    while True:
        if math.gcd(e, fi) == 1:
            break
        e += 1
    public_key = e

    #calculating d using Euclidean algorithm
    gcd, d, x = egcd(e,fi)
    private_key = d%fi
```

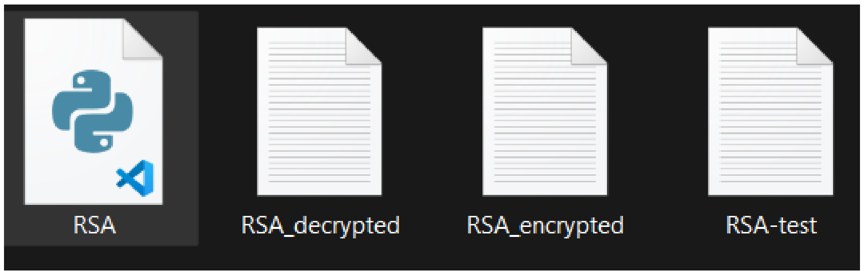
The encryption() function will take the plaintext and by converting each character to unicord and then calculate with the use of modular exponentiation the cipher text is obtained.

The decryption() function will take the cipher text that was just encrypted and will take each character and do the modular exponentiation to get the plain text back.



```python
# Function to encrypt plaintext
def encryption(plaintext):
    e = public_key
    #convert each character to unciord and calculate modular exponention
    ciphertext = [modularExponentiation(ord(char), e, n) for char in plaintext]
    return ciphertext

# Function to decrypt ciphertext
def decryption(ciphertext):
    d = private_key
    #convert each character back to string and calculate using modlar exponention
    plaintext = ''.join([chr(modularExponentiation(char, d, n)) for char in ciphertext])
    return plaintext
```

The program will ask for a file to be encrypted and decrypted which is implemented with error handling. After the user enters the filename, it will read the file and encrypt it onto a file named 'RSA_encrypted.txt'. It will also decrypt the encrypted text onto a file names 'RSA_decrypted.txt'.



The program was implemented successfully with it giving the accurate outputs. The programs were implemented successfully with the help of lecture resources and resources found online, helping to avoid, or overcome difficulties.

THANK YOU