

**Name:** Salah Mahamod

**Student ID:** 20152428

**Unit:** ISEC2000 Fundamental Concepts of Cryptography

**Date:** 22/05/2022

**Assessment:** Assignment 2, weighs 25% of the final mark

## Assignment 2 Report

### Declaration of Originality

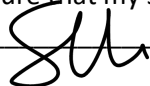
*I declare that:*

- 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:** \_\_\_\_\_ **Date:** 22/05/2022



## Table of Contents

Table of Contents.....	1
Question Answering.....	2
Question 1.....	2
Answer to Question 1 .....	2
Question 2.....	3
Answer to Question 2 .....	3
Programming .....	4
General Overview of my implementation .....	4
Answers to Questions in the Assignment Specification.....	5
References .....	6

## Question Answering

### Question 1

The Euclidean algorithm is based on the following assertion.

Given two integers  $a, b$ , ( $a > b$ ),

$$\gcd(a, b) = \gcd(b, a \bmod b) \quad (1)$$

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

### Answer to Question 1

Firstly,

$$\begin{aligned} \text{if } a = 0, \quad \gcd(0, b) &= b, \\ \text{and, if } b = 0, \quad \gcd(a, 0) &= a \quad (0) \end{aligned}$$

Furthermore, let's assume that  $a$  and  $b$  are both positive integers greater than 0.

$$\text{let, } \gcd(a, b) = d \quad (1)$$

Since  $d$  is the greatest common divisor of  $a$  and  $b$  then this can be rewritten as,

$$a = d \times x \quad (2) \text{ and } b = d \times y \quad (3)$$

If  $a > b$  then,  $x > y$ , also note that  $x$  and  $y$  are coprime.

Secondly,

$$\text{let, } c = a - b \quad (4)$$

Now if the difference of  $a$  and  $b$  is equal to  $c$ , substituting (2) and (3) into (4)

$$c = d \times x - d \times y$$

$$c = d(x - y)$$

$$\text{i.e., } c = (x - y) * \gcd(a, b) \quad (5)$$

This shows that  $d$  is a common divisor of  $a, b$  and  $c$ .

For  $\gcd(b, c)$  note that,  $c = m \times \gcd(b, c)$  (6) and  $b = n \times \gcd(b, c)$  (7),  $m$  and  $n$  are integers.

Since  $a - b = c$ , then  $a = b + c$ , hence  $a = (m + n) \times \gcd(b, c)$  (8)

$\gcd(a, b) \leq \gcd(b, c)$  (9) because  $\gcd(b, c)$  is the greatest common divisor of  $b$  and  $c$ .

Also,  $\gcd(b, c) \leq \gcd(a, b)$  (10) because  $\gcd(a, b)$  is the greatest common divisor of  $a$  and  $b$ .

With (9) and (10) it is only possible if and only if  $\gcd(a, b) = \gcd(b, c)$  (11)

(11) could be rewritten as, substituting (4) in

$$\gcd(a, b) = \gcd(b, a - b) \quad (12)$$

And with (12), you could keep getting the difference of the two integers, till  $a - b = 0$ , and with (0)

$$\gcd(a, b) = \gcd(a, a - b) = \dots = \gcd(d, 0) = d, \quad Q.E.D$$

## Question 2

Assuming that Alice signed a document  $m$  using the RSA signature scheme. (You should describe the RSA signature structure first with a diagram and explain the authentication principle). The signature is sent to Bob. Accidentally Bob found one message  $m'$  ( $m \neq m'$ ) such that  $H(m) = H(m')$ , where  $H()$  is the hash function used in the signature scheme. Describe clearly how Bob can forge a signature of Alice with such  $m'$ .

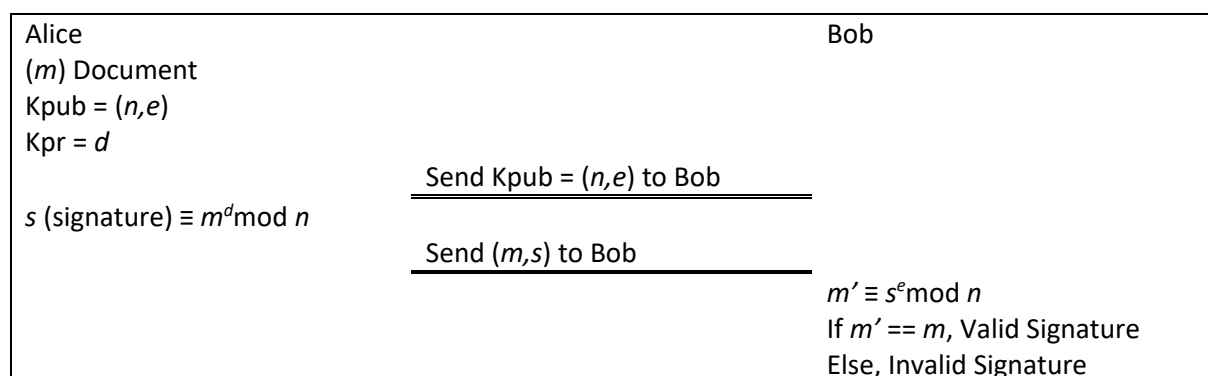
### Answer to Question 2

#### Describe the RSA signature structure with a diagram

The sender of the message will generate their public and private keys ( $K_{pub} = n, e$  and  $K_{pr} = d$ ). The receiver of the message will access the sender's public key ( $n, e$ ) before transmission. This is usually exchanged when a tcp session is established.

The sender will generate a signature for the message ( $m$ ) by using the private key (equation  $signature \equiv m^d \mod n$ ), this signature is appended to the message and sent to the receiver.

The receiver now, verifies the signature by using the public key (equation  $m' = signature^e \mod n$ , where  $m'$  is the message used in the signature NOT necessarily the message sent with it), and checks if the message used in the computation of the signature is the same as the message that was sent.



#### Explain the authentication principle

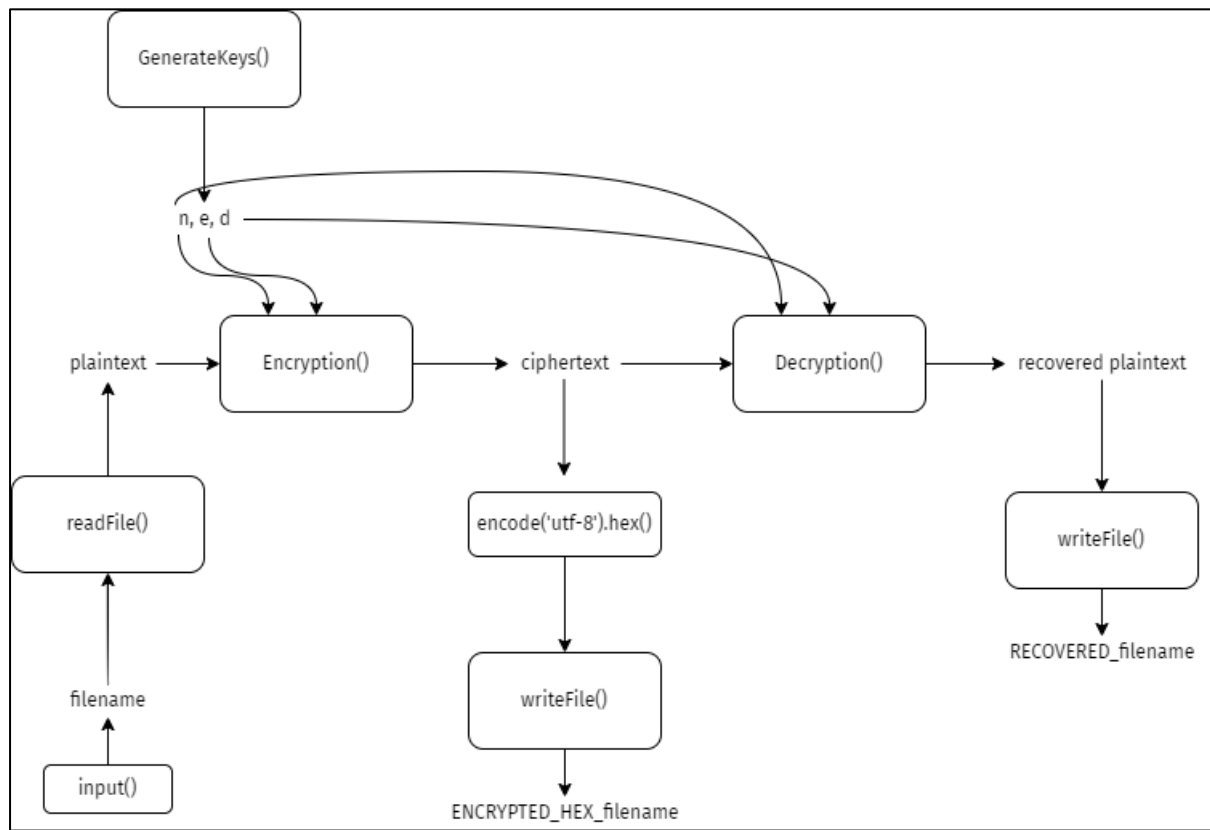
The authentication principle is the guarantee that the sender of a message is authentic, i.e., the message has truly come from the sender. Digital Signatures provide this security service as the original message from the sender is used to create a signature which is later verified

#### Describe how Bob can forge a signature of Alice

Using Alice's public key, Bob can choose a signature ( $s$ ) from the set  $\{0, 1, 2, \dots, n-1\}$ . Bob can then generate message ( $p$ ) from the signature such that,  $p \equiv s^e \mod n$ . Now signature is appended to the message  $p$  ( $p, s$ ). If Bob was to verify this message,  $m' \equiv s^e \mod n$ , then  $m'$  would be equivalent to  $p$  because,  $p \equiv s^e \mod n$ .

# Programming

## General Overview of my implementation



The diagram above illustrates the flow of the program. The user is asked to enter the filename, and then a key pair is generated. The contents of the file are encrypted (in `encryption()`). The resulting ciphertext is written to a file (in hexadecimal) with the prefix “ENCRYPTED\_HEX\_”. The ciphertext is then decrypted (in `decryption()`). The resulting plaintext is written to a file with the prefix “RECOVERED\_”.

The python code contains constants that allow the size of the keys to be adjusted, the current key length is 128 (with  $p$  and  $q$  as 64). The public exponent is set as 65537, as it doesn't have to be secret. The program encrypts blocks of 10 bytes (i.e., characters) at a time – which is specified by another constant.

When generating a prime number, a random number between bounds (this depends on the constants set) is generated. Kehr (2009) argues that it's best to check if the number is even first, then check if it's divisible by the lower primes before using a primality test as it adds reduces overhead. I have used his method of generating large prime numbers (which I have cited), Kehr uses the Miller-Rabin primality test, which I have replaced with the Lehmann primality test I have implemented myself.

## Answers to Questions in the Assignment Specification

*What are the lessons you learned, and difficulties you met, in the process of implementing RSA?*

I have learnt about pseudo-random number generators, how to use them to generate large numbers and how to use them with primality tests such as Lehmann to generate keys for the implementation of RSA.

One of the mistakes I made was accidentally setting the value of  $\phi(n)$  to be equal to  $(p-1)(p-1)$  instead of  $(p-1)(q-1)$ . Although my  $e$  and  $q$  were modular inverses, my  $\phi(n)$  was wrong.  $\phi(n)$  affects what the private exponent is going to be and dictates whether the decryption reverses encryption. It wasn't till I understood this properly that I realised where I've gone wrong.

I also got stuck and wasn't able to find any faults in my code. I decided to use a constant  $n$ ,  $e$  and  $d$  for the keys, use a smaller string as plaintext and print statements to see what values I was getting every step and I easily found my mistake.

*Describe what you have done for source coding and decoding.*

For encryption, I encrypted the plaintext block-by-block (length is specified as a constant) and padded the ciphertext with a space. The problem I faced here was that different blocks would result in different lengths of ciphertexts and using a space made sense, as I could split the ciphertext using space as the delimiter. (i.e., `ciphertext.split(" ")`)

For each block, each character in the plaintext file was converted into its ASCII decimal equivalent. The first character's value was placed at the front (as 3 digits) with the second character's value appended to it. (i.e., "abcd" -> "097098099100"). This string was converted into an integer which was used as the message in the encryption equation. The resulting integer was converted back in reverse (i.e., "101102103097" -> "efga"). For decryption the same mechanism was used.

## References

Paar, Christof. and Pelzl, Jan. 2009. Understanding Cryptography. Leuven: Springer.  
<https://www.crypto-textbook.com/>

Kehrer, Paul. 2009. "Generating (Very) Large Primes." LANGUI.SH.  
<https://langui.sh/2009/03/07/generating-very-large-primes/>

Paar, Christof. 2014. Lecture 11: Number Theory for PKC: Euclidean Algorithm, Euler's Phi Function & Euler's Theorem. YouTube video. 01:31:01. <https://www.youtube.com/watch?v=fq6SXBtUI>

Paar, Christof. 2014. Lecture 12: The RSA Cryptosystem and Efficient Exponentiation by Christof Paar. YouTube video. 01:28:26. <https://www.youtube.com/watch?v=QSIWzKNbKrU>

Paar, Christof. 2014. Lecture 18: Digital Signatures and Security Services by Christof Paar. YouTube video. 01:17:14. <https://www.youtube.com/watch?v=jbBe4AS5pk0&t=349s>

Clay, Adam. 2020. Digital signatures in RSA. YouTube video. 09:02.  
<https://www.youtube.com/watch?v=rLR8WcXy03Q>

Neso Academy. 2021. Extended Euclidean Algorithm (Solved Example 1). YouTube video. 10:15.  
<https://www.youtube.com/watch?v=lq285DDdmtw>