

ISEC2000 Fundamental CONCEPTS OF Cryptography Assignment 2

2025



May 16, 2025

Andrei Vann Anthony A Bedruz

Curtin University

**A paper with text and a black text

AI-generated content may be incorrect.**

Contents

[Part 1: Conceptual Understanding and Question Answering 4](#_Toc198388439)

[1.) Explain the core principle of public key cryptography and how it addresses the issue of secure communication over untrusted channels. 4](#_Toc198388440)

[2.) Given two prime numbers p = 61 and q = 53, and public key exponent e = 17.  Calculate the modulus n, Euler’s totient ϕ(n).  Compute the private key d.  Convert “SECURE” to ASCII and encrypt using RSA. Show the working steps. 4](#_Toc198388441)

[3.) The Euclidean algorithm is based on the following assertion. Given two integers a, b, (a < b), gcd(a, b) = gcd(a, b mod a) 5](#_Toc198388442)

[a). (1) Prove the assertion (1) mathematically. (Note that proof by example is NOT appropriate here) 5](#_Toc198388443)

[4.) You are given the following parameters for a simplified Diffie-Hellman key exchange:  6](#_Toc198388444)

[ Prime modulus p = 467, Generator g = 2, Alice’s private key a = 153, Bob’s private key b = 197 6](#_Toc198388445)

[Now, 6](#_Toc198388446)

[a. Please compute both the public keys and the shared key. 6](#_Toc198388447)

[b. Describe a possible Man-in-the-Middle (MitM) attack scenario during the key exchange between Alice and Bob. Include the attacker’s steps and how the attack compromises security 6](#_Toc198388448)

[5.) Answer the following questions: 7](#_Toc198388449)

[a. Explain how Elliptic Curve Cryptography (ECC) achieves 7](#_Toc198388450)

[security with smaller key sizes than RSA. 7](#_Toc198388451)

[b. Compare RSA-2048 vs ECC-256 in terms of speed and memory. 7](#_Toc198388452)

[c. If you are designing encryption for IoT devices, would you 7](#_Toc198388453)

[choose ECC or RSA? Justify with technical reasoning. 7](#_Toc198388454)

[Part 2 8](#_Toc198388455)

[Steps to RSA Algorithm: 8](#_Toc198388456)

[1.) Create two prime numbers: 8](#_Toc198388457)

[2.) Generate the keys: 9](#_Toc198388458)

[3.) Encryption Process 10](#_Toc198388459)

[4.) Decryption process 10](#_Toc198388460)

[5.) Signature process 11](#_Toc198388461)

[6.) Signature verification process 11](#_Toc198388462)

[7.) Test File 12](#_Toc198388463)

[Conclusion 12](#_Toc198388464)

[References 13](#_Toc198388465)

# Part 1: Conceptual Understanding and Question Answering

## Explain the core principle of public key cryptography and how it addresses the issue of secure communication over untrusted channels.

Public key cryptography uses two keys to encrypt and decrypt data, one being a public key that is shared publicly while the other is a private key that is kept a secret by the recipients of data. When information is shared through untrusted channels, they are already encrypted with a public key. Once this information is received, it is then decrypted with the private key, which the recipient is the only person who has access to this key. Since only the recipient has access to this key, it cannot be intercepted by foreign means, even by the sender. This ensures confidentiality, allowing only one person access to the private key.

## Given two prime numbers p = 61 and q = 53, and public key exponent e = 17.  Calculate the modulus n, Euler’s totient ϕ(n).  Compute the private key d.  Convert “SECURE” to ASCII and encrypt using RSA. Show the working steps.

A white paper with black text

AI-generated content may be incorrect.A white paper with black text

AI-generated content may be incorrect.find the modulus using two prime numbers.
A white board with black text

AI-generated content may be incorrect.Modulus (n) = 3233 Euler’s Totient ϕ(n) = 3120

Private key (d) = 2753

Convert SECURE and encrypt using RSA:[2680, 28, 641, 2310, 1859, 28]

## The Euclidean algorithm is based on the following assertion. Given two integers a, b, (a < b), gcd(a, b) = gcd(a, b mod a)

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

We know from above:

r=b−q\*a

Now, if a number divides both a and b, then it must also divide any integer combination of a and b.

Since r is equal to b−q\*a, and both b and a are divisible by the common factor, r must also be divisible by that same factor.

So:

* Every number that evenly divides both a and b also divides r
* Therefore, the set of common divisors of a and b is included in the set of common divisors of a and r

We use the same identity for the reverse:

b=q\*a+r

If a number divides both a and r, then it must divide any integer combination of them — including q\*a+ r, which is b.

So:

* Every number that evenly divides both a and r also divides b
* Therefore, the set of common divisors of a and r is included in the set of common divisors of a and b

## You are given the following parameters for a simplified Diffie-Hellman key exchange: 

## Prime modulus p = 467, Generator g = 2, Alice’s private key a = 153, Bob’s private key b = 197

## Now,

## A white board with black text AI-generated content may be incorrect.Please compute both the public keys and the shared key.

Alice public key = 224, Bob public key = 87, shared key = 367

## b. Describe a possible Man-in-the-Middle (MitM) attack scenario during the key exchange between Alice and Bob. Include the attacker’s steps and how the attack compromises security

How the key exchange happens:

* Alice sends her public key (A = g^amodp) to Bob.
* Attacker intercepts it, does not forward it to Bob. The attacker creates her own private key and generate a public key to be sent to Bob (X1 =g^x1modp).
* Bob thinking this is Alice, sends his public key (B = b^bmodp) as well to Alice.
* It is intercepted again by the attacker and creates another private key to generate a public key to be sent back to Alice(X2 = g^x1modp).

Now the attacker has both the public keys of each sender, without the senders knowing it was intercepted. This is a breach of confidentiality as attacker can see all the messages, Integrity as attacker can modify messages, and Authenticity as both Alice and Bob did not authenticate with each other.

The attacker can find the shared key one private key at a time, by the formula

K1 = X2^amodp = g^(x2a)modp

K2 = X1^bmodp = g^(x1b)modp

## 5.) Answer the following questions:

## a. Explain how Elliptic Curve Cryptography (ECC) achieves

## security with smaller key sizes than RSA.

The main reason how ECC achieves security with smaller key sizes is due to the complexity and difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP). Because of the logarithm of elliptic curves , it is more computationally expensive for a given key length. Comparing it to RSA where it just uses large prime numbers, ECC can also use negative numbers, as it is grid based, adding more complexity.

## b. Compare RSA-2048 vs ECC-256 in terms of speed and memory.

|  |  |  |
| --- | --- | --- |
|  | RSA- 2048 | ECC-256 |
| Speed | Key Generation is a lot bigger therefore slower than ECC. Its Encryption is much faster as there’s less complexity involved. But its decryption process is much slower. | Smaller key size means less key generation. Encryption is slower but not as significant compared to RSA. Decryption and signature generation is a lot faster though. |
| Memory | Requires more memory due to the larger key size and larger signatures. | Uses less memory due to a significantly lower key size without sacrificing security. |

## c. If you are designing encryption for IoT devices, would you

## choose ECC or RSA? Justify with technical reasoning.

ECC is much more secure without risking performance and security. It is more efficient due to demanding lower memory and has better key generation due to not needing to compute larger key and signatures. It is also used in a lot of security protocols, therefore easily scalable and integrable with other systems.

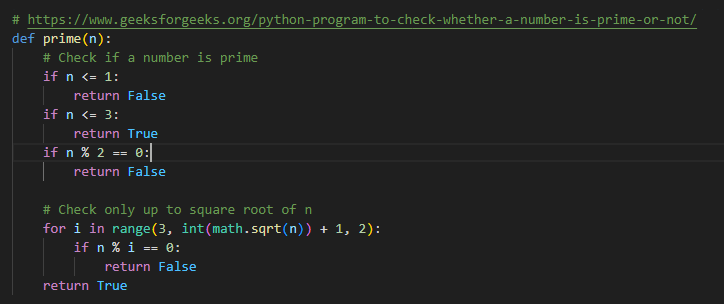
# Part 2

Create 2048-bit RSA with a public exponent of 65537 saving the private and public keys as ‘private.pem’ and ‘public.pem’ respectively. Ensure that the confidential\_message.txt text file can be encrypted and decrypted with the signature using SHA-256. Verify the signature using the public key, ensuring verification process is successful.

## Steps to RSA Algorithm:

### Create two prime numbers:

First thing we must do is generate two prime numbers. We must ensure that these numbers are prime and are generated randomly.

To ensure primality, we can use a for loop to check from a range from 2 to designated number (n) to see if it is divisible by something. There are more efficient methods to this, but it gets the job done and it gets the job done.The *generate\_prime* function just uses the *random* module to get a random number of bits and checking if its prime. These are then called to generate our two prime numbers in the *encrypt\_document* function.

**Prime function that goes through the loop from one at a time from 2 to square root of n +1(lines 5 – 19)**

When testing happened with the keys, this approach was far too inefficient as it goes through square root of n + 1 times max. I had to replace it with something more efficient approach.

The Miller – Rabin Primality test was a more efficient approach, especially with larger numbers. It does multiple rounds of testing, verifying each round if a randomly chosen base witnesses the compositeness of the number.

A computer screen shot of a program

AI-generated content may be incorrect.The approach is much faster than the first iteration.

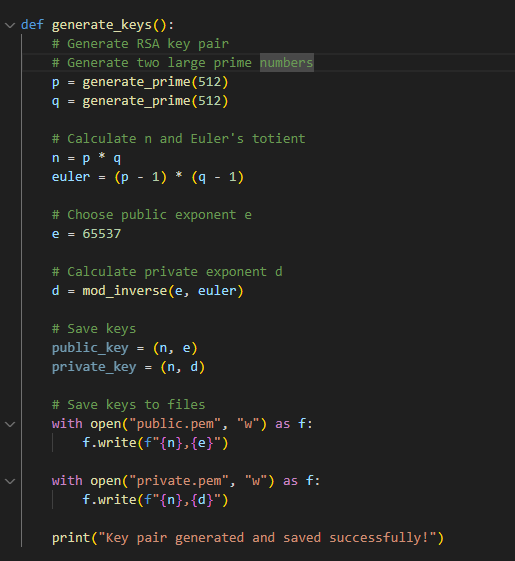
**A more efficient method of getting a big prime number; the Miller - Rabbin method. (lines 22 – 49)**

### Generate the keys:

We can now generate the two prime numbers to generate the keys and calculate Euler’s Totient and the Modulus (n) with the formulas:

n = p \* q

euler = (p-1)(q-1)

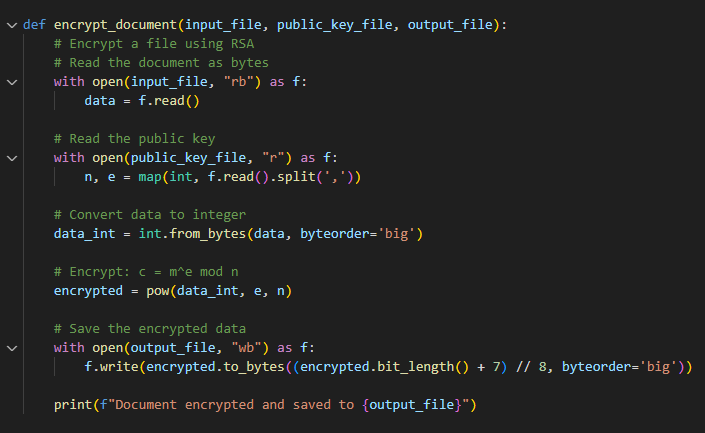
We can then calculate the private key using the mod inverse of the *public exponent(*65537) and *euler.* We can store the public exponent and the modulus to public.pem, as well as the private exponent and the modulus to private.pem .

**Generate\_keys() function generates the prime number, calculates the keys exponents, and then store them into their designated files of public.pem or private.pem respectively (lines 73 – 100)**

### Encryption Process

The encryption process goes through the following steps:

* Open and read the confidential\_message.txt file as bytes. Saved these bytes into a variable (*data)*
* Convert the data variable into an integer for computation purposes.
* Open public.pem file and store the modulus and the public exponent as integers by comma- splitting the file itself.
* Encrypt the data with the encryption formula of *c = m^e mod n*  where m is the data variable. Save c as a variable as well (*encrypted)*
* Save the encrypted data into a separate file.



**Encrypt\_document function reads and encrypts the file by using the public.pem that was stored with the generate\_keys function. The encrypted bytes are then stored into an output\_file (lines 102 – 123)**

### Decryption process

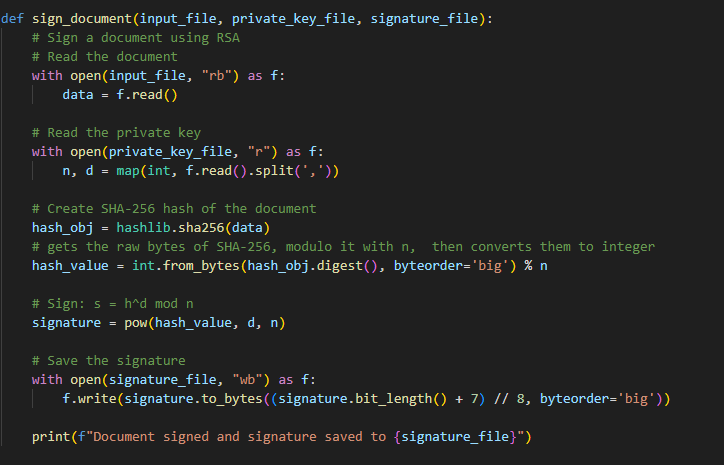
The decryption process is similar to the encryption process, as it only changes what files to open (the output\_file from the encrypt\_document function and the private.pem). The extra step of converting the bytes to string is another step in the process as we need to convert it from integer to bytes and then decoded back into a string.

A screen shot of a computer program

AI-generated content may be incorrect.

**decrypt\_document function reads the encrypted file, opens private.pem to read the modulus and the private exponent (lines 102 – 123)**

### Signature process

The signature process reopens the encrypted message, uses the SHA-256 from the hashlib library to create the hash value that will be included in the signature equation (s = h^d mod n, where h is the hash value using the modulus and d is the private exponent). We then save the signature into a signature\_file, ensuring the data is stored into bytes.

**Sign\_document function reads the data file, opens private.pem to read the modulus and the private exponent, create a hash value using SHA- 256 and then uses this to create a signature that can then be stored into a signature\_file(lines 146 – 168)**

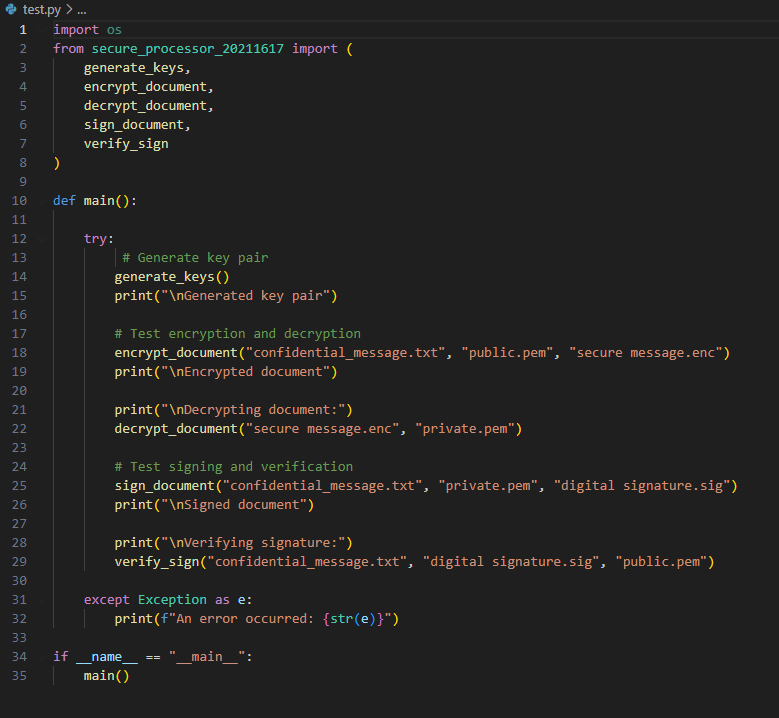
### Signature verification process

The final process is to verify and authenticate that the file is signed properly. We can do this by opening the file again, now using the public key the same way as creating the signature, finding a hash value of the data SHA – 256 algorithms, modulo it with n and then and then comparing it with the signature equation. if they are the same, then the signature verification is successful.

**Verify\_signfunction reads the data file, opens public.pem to read the modulus and the public exponent, and then compares the hash value from the modulus and the data to the signature equation of h = s^e mod n. If they are the same, then the signature is successful. (lines 170 – 195)**

### Test File

The test file shows the implementation of the RSA algorithm. It first generates the key pairs. Then encrypt the file using encrypt\_document. It then uses decrypt\_document to decrypt the file. Lastly, it will include a signature file and verify it.



# Conclusion

The RSA program implements prime number generation, key generation, encryption and decryption as well as inclusion and verification of digital signatures. It has demonstrated the practicality and security of RSA, as well as its limitations. The large number of computational resources needed to generate and use large key sizes shows how it could slow at the beginning, but with its fairly simple encryption and decryption process shows its efficiency.

Overall, this has deepened my understanding of the RSA algorithm and provided valuable insight and experience with its security – focused features.

# References

GeeksforGeeks. (n.d.). *Public key encryption*. <https://www.geeksforgeeks.org/public-key-encryption/>

GeeksforGeeks. (n.d.). *Euler’s totient function*. <https://www.geeksforgeeks.org/eulers-totient-function/>

GeeksforGeeks. (n.d.). *RSA algorithm (cryptography)*. <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>

GeeksforGeeks. (n.d.). *Python program to check whether a number is prime or not*. <https://www.geeksforgeeks.org/python-program-to-check-whether-a-number-is-prime-or-not/>

Cryptography.io. (n.d.). *RSA — cryptography.hazmat.primitives.asymmetric.rsa*. <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/#module-cryptography.hazmat.primitives.asymmetric.rsa>

Lecture 6. (n.d.). *Public key cryptography - RSA* [PDF file]. Blackboard.

Lecture 7. (n.d.). *Diffie-Hellman key exchange* [PDF file]. Blackboard.

Khan Academy. (2020, September 15). *Elliptic curve cryptography: an overview* [Video]. YouTube. <https://www.youtube.com/watch?v=rzSU2m8oN48>

Up and Atom. (2019, March 13). *Elliptic curves: the hard math behind Bitcoin* [Video]. YouTube. <https://www.youtube.com/watch?v=NF1pwjL9-DE>

Computerphile. (2017, November 2). *RSA encryption: The math of the RSA algorithm* [Video]. YouTube. <https://www.youtube.com/watch?v=qph77bTKJTM>

Python Software Foundation. (n.d.). *hashlib — Secure hashes and message digests*. Python 3.12.3 documentation. Retrieved May 17, 2025, from <https://docs.python.org/3/library/hashlib.html#hashlib.sha256>