Course Name:	Applied Cryptography	Semester:	V
Date of Performance:	11 / 08 / 2025	DIV/ Batch No:	D2
Student Name:	Shreyans Tatiya	Roll No:	16010123325

Experiment No:5

Title: Understanding Asymmetric Key Cryptography Algorithms

Aim and Objective of the Experiment:

Implementation of RSA algorithm and understanding RSA cryptanalysis

COs to be achieved:

CO2: Demonstrate and implement various Cryptographic Algorithms for securing systems.

Books/ Journals/ Websites referred:

1. Stallings, W., Cryptography and Network Security: Principles and Practice, Second edition, Person Education
2. Forouzan, B. A. (2018). Cryptography and Network Security. McGraw-Hill Education.

Theory: Explain the following.

Explain the requirement of asymmetric key cryptography**(students are supposed to explain)**:

In symmetric key cryptography, the same secret key is used by both the sender and the receiver to encrypt and decrypt messages. While this works well in small, controlled environments, it quickly runs into two big problems. First, there's the key distribution problem — you have to share the secret key with the other person before you can talk securely, but if someone intercepts that key during exchange, the whole system is broken. Second, it doesn't scale well. In a network of many users, each pair of people would need their own unique shared key, and the number of keys grows rapidly as the network grows.

Asymmetric key cryptography solves these problems in a very clever way. Instead of one shared key, it uses a pair of keys — a public key and a private key. The public key can be given to anyone, and it's used for encrypting messages or verifying digital signatures. The private key is kept secret

by its owner, and it's used for decrypting messages or creating signatures. Since only the public key needs to be shared, there's no need for a secure channel just to exchange keys.

This system also makes it possible to do more than just keep messages secret. With asymmetric cryptography, a sender can "sign" a message with their private key so that anyone can verify it with the matching public key. This not only proves that the message came from the sender but also ensures it hasn't been tampered with. In short, asymmetric cryptography is essential for modern communication because it removes the headaches of key exchange, works well even in large open networks, and provides both confidentiality and trust.

RSA(Rivest-Shamir-Adleman) Algorithm is an asymmetric or public-key cryptography algorithm which means it works on two different keys: Public Key and Private Key. The Public Key is used for encryption and is known to everyone, while the Private Key is used for decryption and must be kept secret by the receiver. RSA Algorithm is named after Ron Rivest, Adi Shamir and Leonard Adleman, who published the algorithm in 1977.

RSA Algorithm is based on factorization of large number and modular arithmetic for encrypting and decrypting data. It consists of three main stages:

1. Key Generation: Creating Public and Private Keys
2. Encryption: Sender encrypts the data using Public Key to get cipher text.
3. Decryption: Decrypting the cipher text using Private Key to get the original data.

1. Key Generation

- Choose two large prime numbers, say p and q. These prime numbers should be kept secret.
- Calculate the product of primes, n = p * q. This product is part of the public as well as the private key.

- Calculate Euler's Totien Function

  $\Phi(n)$ as $\Phi(n) = \Phi(p * q) = \Phi(p) * \Phi(q) = (p - 1) * (q - 1)$.
- Choose encryption exponent e, such that
  - $1 < e < \Phi(n)$, and
  - $\gcd(e, \Phi(n)) = 1$, that is e should be co-prime with $\Phi(n)$.
- Calculate decryption exponent d, such that
  - $(d * e) \equiv 1 \bmod \Phi(n)$, d is modular multiplicative inverse of e mod $\Phi(n)$. Some common methods to calculate multiplicative inverse are the extended Euclidean Algorithm, Fermat's Little Theorem, etc.
  - We can have multiple values of d satisfying $(d * e) \equiv 1 \bmod \Phi(n)$ but it does not matter which value we choose as all of them are valid keys and will result in same message on decryption.
  - Finally, the Public Key = (n, e) and the Private Key = (n, d).

2. Encryption: To encrypt a message M, it is first converted to numerical representation using ASCII and other encoding schemes. Now, use the public key (n, e) to encrypt the message and get the cipher text using the formula:

$C = M^e \bmod n$, where C is the Cipher text and e and n are parts of public key.

3. Decryption

To decrypt the cipher text C, use the private key (n, d) and get the original data using the formula:

$M = C^d \mod n$, where M is the message and d and n are parts of private key.

RSA implementation Code and Output :

```
import math
from sympy import mod_inverse, factorint, isprime

p = 61
q = 53
n = p * q
phi = (p-1)*(q-1)
e = 17
d = mod_inverse(e, phi)

print(f"Public Key (n, e): ({n}, {e})")
print(f"Private key (n, d): ({n}, {d})")

M = 23
print(f"\nOriginal message M = {M}")

sign = pow(M, d, n)
print(f"Signature (M^d mod n) = {sign}")

rec = pow(sign, e, n)
print(f"Recovered message (signature^e mod n) = {rec}")
```

```
(venv) PS D:\KJSCE\SEM-5\AC\Lab\Expt-3> python -u "d:\KJSCE\SEM-5\AC\Lab\Expt-3\RSA.py"
Public Key (n, e): (3233, 17)
Private key (n, d): (3233, 2753)

Original message M = 23
Signature (M^d mod n) = 765
Recovered message (signature^e mod n) = 23
```

RSA Cryptanalysis Code and Output :

```
import math
from sympy import mod_inverse
```

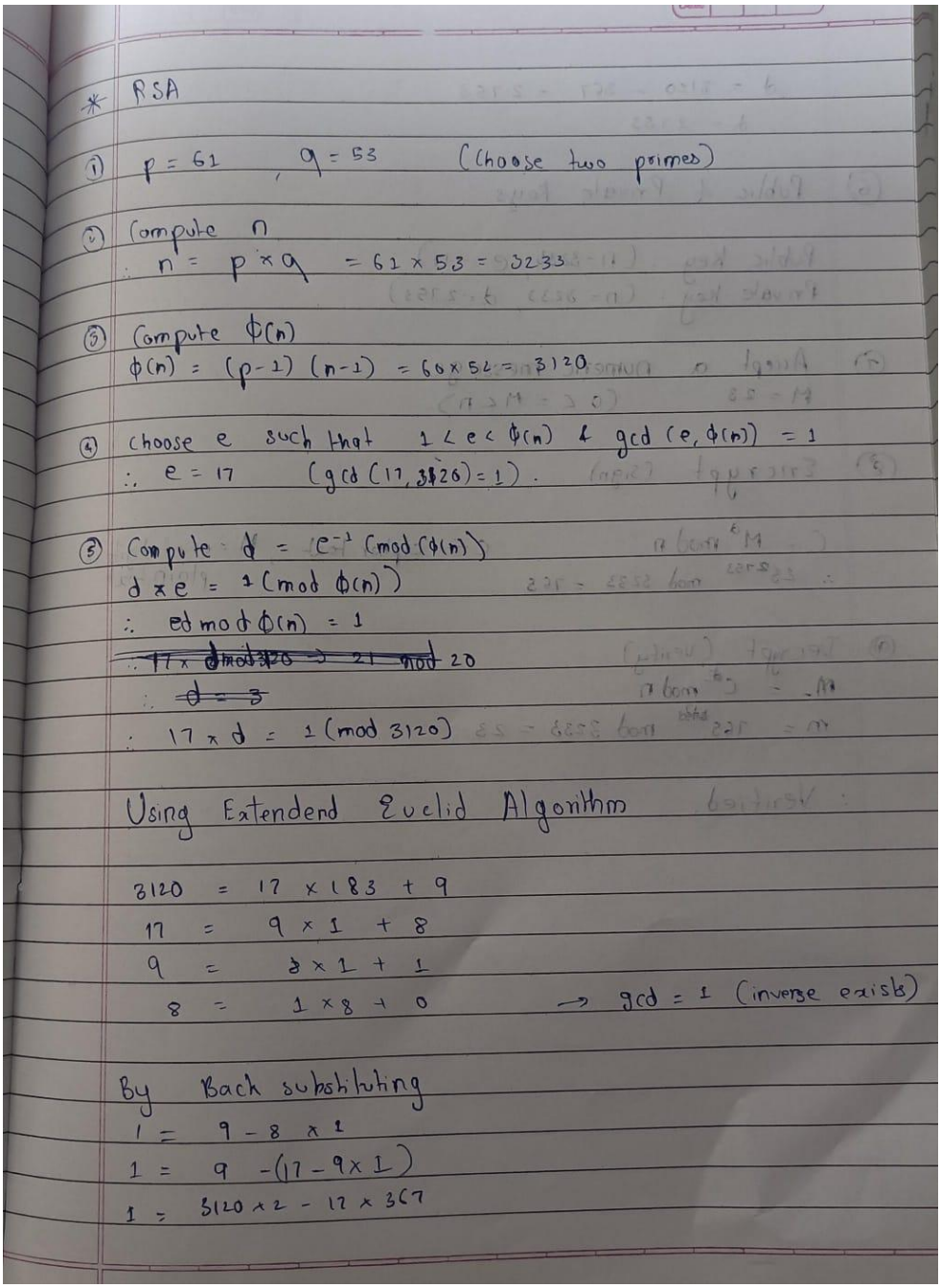
```python
n = 3233
e = 17
sign = 765

def factor_trial(n):
    if n % 2 == 0:
        return 2, n // 2
    limit = int(math.isqrt(n)) + 1
    for i in range(3, limit, 2):
        if n % i == 0:
            return i, n // i
    return None, None

p, q = factor_trial(n)
print(f"Factors found: p = {p}, q = {q}")

phi = (p - 1) * (q - 1)
print(f"Computed phi(n) = {phi}")

d = mod_inverse(e, phi)
print(f"Computed private key d = {d}")

rec = pow(sign, e, n)
print(f"Recovered message from signature^d mod n = {rec}")
```

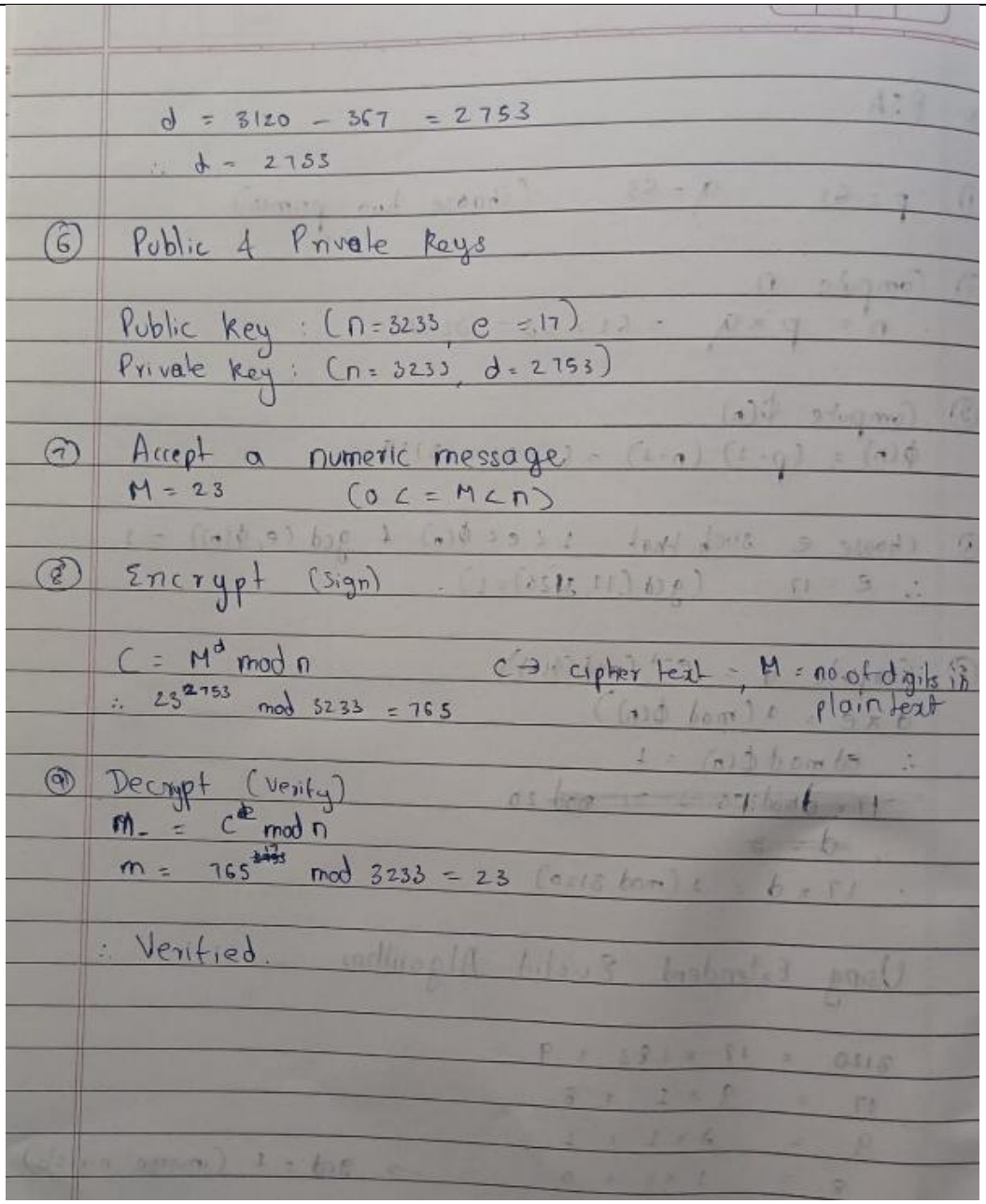
```
(venv) PS D:\KJSCE\SEM-5\AC\Lab\Expt-3> python -u "d:\KJSCE\SEM-5\AC\Lab\Expt-3\RSA.py"
Factors found: p = 53, q = 61
Computed phi(n) = 3120
Computed private key d = 2753
Recovered message from signature^d mod n = 23
```

Post Lab Subjective/Objective type Questions:

1. Solve one example using RSA algorithm.

Ans:





$d = 3120 - 367 = 2753$

$\therefore d = 2753$

(6) Public & Private Keys

Public Key : $(n = 3233, e = 17)$
Private Key : $(n = 3233, d = 2753)$

(7) Accept a numeric message
$M = 23 \quad (0 < = M < n)$

(8) Encrypt (Sign)

$C = M^d \bmod n$ C → cipher text, M = no of digits in plain text

$\therefore 23^{2753} \bmod 3233 = 765$

(9) Decrypt (Verify)

$M = C^e \bmod n$

$m = 765^{17} \bmod 3233 = 23$

$\therefore$ Verified.

2. Write Applications of RSA.

Ans:

- Secure Communication: RSA is widely used to keep online conversations private. When you visit websites starting with "https," RSA helps establish a secure connection so that hackers can't read your data.
- Digital Signatures: It allows people to sign documents or emails electronically in a way that proves the message really came from them and wasn't changed along the way. This builds trust in digital transactions.
- Email Encryption: RSA helps encrypt emails, making sure that only the intended recipient can read the message, keeping sensitive information safe.
- Software Security: Developers use RSA to sign software and updates, so users can verify the software is genuine and hasn't been tampered with.
- Authentication: Many systems use RSA as part of login processes to verify user identities without sending passwords over the network, making logins safer.

3. Comment on the strengths and weaknesses of RSA.

Ans:

Strengths of RSA:

- RSA is one of the earliest and most widely used encryption methods, so it's well understood and trusted.
- It provides strong security based on the difficulty of factoring large numbers — something computers still struggle with.
- Because it's an asymmetric system, it allows secure communication without sharing secret keys in advance, which is a big advantage.
- RSA also supports digital signatures, which help prove authenticity and integrity of messages or software.

Weaknesses of RSA:

- RSA can be slow compared to newer algorithms, especially when dealing with large

amounts of data, so it's often combined with faster symmetric encryption for actual data transfer.

- If the key size is too small, RSA can be broken by factoring, so it requires very large keys (2048 bits or more) to stay secure.
- It's vulnerable if the implementation isn't done carefully — like using poor random number generators or skipping proper padding schemes.
- Quantum computers, once they become powerful enough, could break RSA by efficiently factoring large numbers, so the long-term future of RSA is uncertain.

Conclusion:

In this project, we implemented the RSA algorithm to generate keys, sign messages, and verify signatures. We also explored RSA cryptanalysis by performing a factorization attack to show how the private key can be recovered when the modulus is small. This helped us understand both the strength and vulnerabilities of RSA encryption.