Terna Engineering College

Computer Engineering Department

Class: TE **Sem.: VI**

Course: System Security Lab

PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.02

A.1 Aim:

Write a program to implement RSA algorithm and Digital Signature scheme using RSA / ElGamal.

A.2 Prerequisite:

1. Basic Knowledge of Asymmetric Key Cryptography.

A.3 Outcome:

After successful completion of this experiment students will be able to;

1. Analyze the various public key cryptographic techniques and their applications.

A.4 Theory:

A. RSA Algorithm:

- **RSA** (**Rivest–Shamir–Adleman**) is one of the first public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and it is different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem". The acronym RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978. Clifford Cocks, an English mathematician working for the British intelligence agency Government Communications Headquarters (GCHQ), had developed an equivalent system in 1973, but this was not declassified until 1997.

- A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret.

Anyone can use the public key to encrypt a message, but with currently published methods, and if the public key is large enough, only someone with knowledge of the prime numbers can decode the message feasibly.[2] Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem remains an open question.

- RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

▪ ALGORITHM:

1. Accept two prime numbers from user (say p and q).
2. Calculate n = p * q.
3. Calculate Φ(n) as

   Φ (n) = (p - 1) * (q – 1).
4. Considering e * d = Φ(n) + 1, determine e and d where e and d are prime numbers.
5. So display information at sender as (e, n) and information at receiver as (d , n).
6. Check whether user is sender or receiver.
7. If user is sender
   a. Get message M from user.
   b. $C = M^e \bmod n$.
   c. Send cipher text C to receiver.
   d. Go to stop.
8. If user is receiver
   a. Get cipher text C from user.
   b. $M = C^d \bmod n$.
   c. Display plain text M to receiver.
   d. Go to stop.
9. Ask whether user wants to continue (yes or no?)

If yes, go to step 7.

Else go to stop.

▪ EXAMPLE:

- Choose p = 3 and q = 11
- Compute n = p * q = 3 * 11 = 33
- Compute φ(n) = (p - 1) * (q - 1) = 2 * 10 = 20
- Choose e such that 1 < e < φ(n) and e and n are co-prime. Let e = 7
- Compute a value for d such that (d * e) % φ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1]
- Public key is (e, n) => (7, 33)
- Private key is (d, n) => (3, 33)
- The encryption of m = 2 is c = 27 % 33 = 29
- The decryption of c = 29 is m = 293 % 33 = 2

B. Digital Signature RSA :

Traditionally signature with a message is used to give evidence of **identity** and **intention** with regard to that message. For years people have been using various types of signature to associate their identity and intention to the messages. Wax imprint, seal, and handwritten signature are the common examples. But when someone need to sign a digital message, things turn different. In case of signing a digital document one cannot use any classical approach of signing, because it can be forged easily. Forger just need to cut the signature and paste it with any other message. For signing a digital document one uses **digital signature**.

Therefore, digital signature are required not to be separated from the message and attached to another. That is a digital signature is required to be both message and signer dependent. For validating the signature anyone can verify the signature, so digital signature is supposed to be verified easily.

A digital signature scheme typically consists of three distinct steps:

1. **Key generation:-** User compute their public key and corresponding private key.
2. **Signing:-** In this step user sign a given message with his/her private key.
3. **Verification:-** In this step user verify a signature for given message and public key.

So the functionality provided by digital signature can be stated as follows:

Authentication:- Digital signature provides authentication of the source of the messages as a message is signed by the private key of the sender which is only known to him/her. Authentication is highly desirable in many applications.

Integrity:- Digital signature provides integrity as digital signature uniquely associate with corresponding message. i.e. After signing a message a message cannot be altered if someone do it will invalidate the signature. There is no efficient method to change message and its signature to produce a new message and valid signature without having private key. So both sender and receiver don't have to worry about in transit alteration.

Non- repudiation:- For a valid signature sender of message cannot deny having signed it.

RSA digital signature scheme

Suppose Alice want to send a *message**(m)** to* Bob. She can generate digital signature using RSA digital signature scheme [4] as follow:

Key Generation:-
She can generate key for RSA signature scheme:

1. Choose two distinct large prime numbers p and q.
2. Compute $n = pq$.
3. n is used as the modulus for both the public and private keys.
4. Compute $\varphi(n) = (p - 1)(q - 1)$, where φ is Euler's totient function.
5. Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$.
6. Compute $d = e-1 \bmod \varphi(n)$.

Then the public key and private key of user will be ***(e, n)*** and ***(d, n)*** respectively.
Now she have her public and private key. Now she can generate the signature of a message by encrypting it by her private key.
So she can generate signature corresponding to message***(m)*** as follow:

Signing: -

1. Represent the message ***m*** as an integer between ***0*** and ***n − 1***.
2. Sign message by raising it to the ***dth*** power modulo ***n***.

$$S \equiv m^d \pmod n$$

So ***S*** is the signature corresponding to message m. Now she can send message ***m*** along with the signature ***S*** to Bob.

Upon receiving the message and signature ***(m, S)***, Bob can verify the signature by decrypting it by Alice public key as follow:

Verification:-

1. Verify signature by raising it to the ***eth*** power modulo ***n***.

   $$m' \equiv S^e \pmod n$$

2. If ***m′ = m (mod n)*** then signature is valid otherwise not.

For a valid signature both ***m*** and ***m′*** will be equal because:
$S \equiv m^d \pmod n$
$m' \equiv m^{de} \pmod n$
and
e is inverse of ***d***, i.e. ***ed ≡ 1(mod Φ(n))***.

So, by using above algorithm Alice can generate a valid signature ***S*** for her message ***m***, but there is a problem in above define scheme that is the length of the signature is equal to the length of the message. This is a disadvantage when message is long.

Additional instructions for RSA signature algorithm is as follows:

An RSA digital signature key pair consists of an RSA private key, which is used to compute a digital signature, and an RSA public key, which is used to verify a digital signature. An RSA digital signature key pair shall not be used for other purposes (e.g. key establishment).

An RSA public key consists of a modulus *n*, which is the product of two positive prime integers ***p***and ***q*** (i.e., ***n = pq***), and a public key exponent ***e***. Thus, the RSA public key is the pair of values ***(n, e)*** and is used to verify digital signatures. The size of an RSA key pair is commonly considered to be the length of the modulus ***n*** in bits (***nlen***). The corresponding RSA private key consists of the same modulus ***n*** and a private key exponent ***d*** that depends on ***n*** and the public key exponent ***e***. Thus, the RSA private key is the pair of values ***(n, d)*** and is used to generate digital signatures. In order to provide security for the digital signature process, the two integers ***p*** and ***q***, and the private key exponent ***d*** shall be kept secret. The modulus n and the public key exponent ***e*** may be made known to anyone.

The Standard specifies three choices for the length of the modulus (i.e., *nlen*): 1024, 2048 and 3072 bits.

An approved hash function, shall be used during the generation of key pairs and digital signatures. When used during the generation of an RSA key pair, the length in bits of the hash function output block shall meet or exceed the security strength associated with the bit length of the modulus ***n***. The security strength associated with the RSA digital signature process is no greater than the minimum of the security strength associated with the bit length of the modulus and the security strength of the hash function that is employed. Both the security strength of the hash function used and the security strength associated with the bit length of the modulus ***n*** shall meet or exceed the security strength required for the digital signature process.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll No.B30	**Name: Pranjal Bhatt**
Class :TE B COMPS	**Batch :B2**
Date of Experiment:	**Date of Submission**
Grade :	

B. Output of RSA Algorithm with Digital Signature :

1

```
--- RSA Key Generation ---
Enter a prime number p:
3
Enter a prime number q:
11
Enter public exponent e (1 < e < 20 and gcd(e, 20) = 1):
7

Public Key (e, n): (7, 33)
Private Key (d, n): (3, 33)

--- Message Encryption & Decryption ---
Enter a message to encrypt (integer less than n):
2
Encrypted Message (Ciphertext): 29
Decrypted Message: 2

--- Digital Signature ---
Enter a message to sign:
mahesh
Digital Signature: 19
Signature Verification: Valid
```

Date : / /
Page No. :

Exp - 1

Q) $P=3$, $q=11$, $e=7$, message $=2$

Step 1 : RSA key Generation

- Prime number $P=3$, $q=11$
- $n = P \times q = 3 \times 11 = 33$
- $\phi(n) = (P-1)(q-1) = 2 \times 10 = 20$

choose public exponent $e=7$ (such that $1 < e < 20$)

$e\,d \bmod \phi(n) = 1$

$\therefore 7d \bmod 20 = 1$

$d = 3$

- Public key $= (R=7, n=33)$
- Private key $= (d=3, n=33)$

Step 2 : Message Encryption & Decryption

Encryption

$m = 2$

$C = m^e \pmod n$

$C = 2^7 \bmod 33 = 29$

Decryption :

$m = C^d \pmod n$

$m = 29^3 \pmod{33} = 2$

step 3 : Digital signature

Message to send = Pranjal

- Digital signature $= d=3$

signature $= m^d$ sined $(\bmod n) = 19^3 \pmod{33}$

$= 19$

- Signature verification : $e=7$

Verification $=$ signature$^e \pmod n$

$= 19^7 \pmod{33} = 19$

signal verification

B. Source Code of RSA Algorithm with Digital Signature:

2.

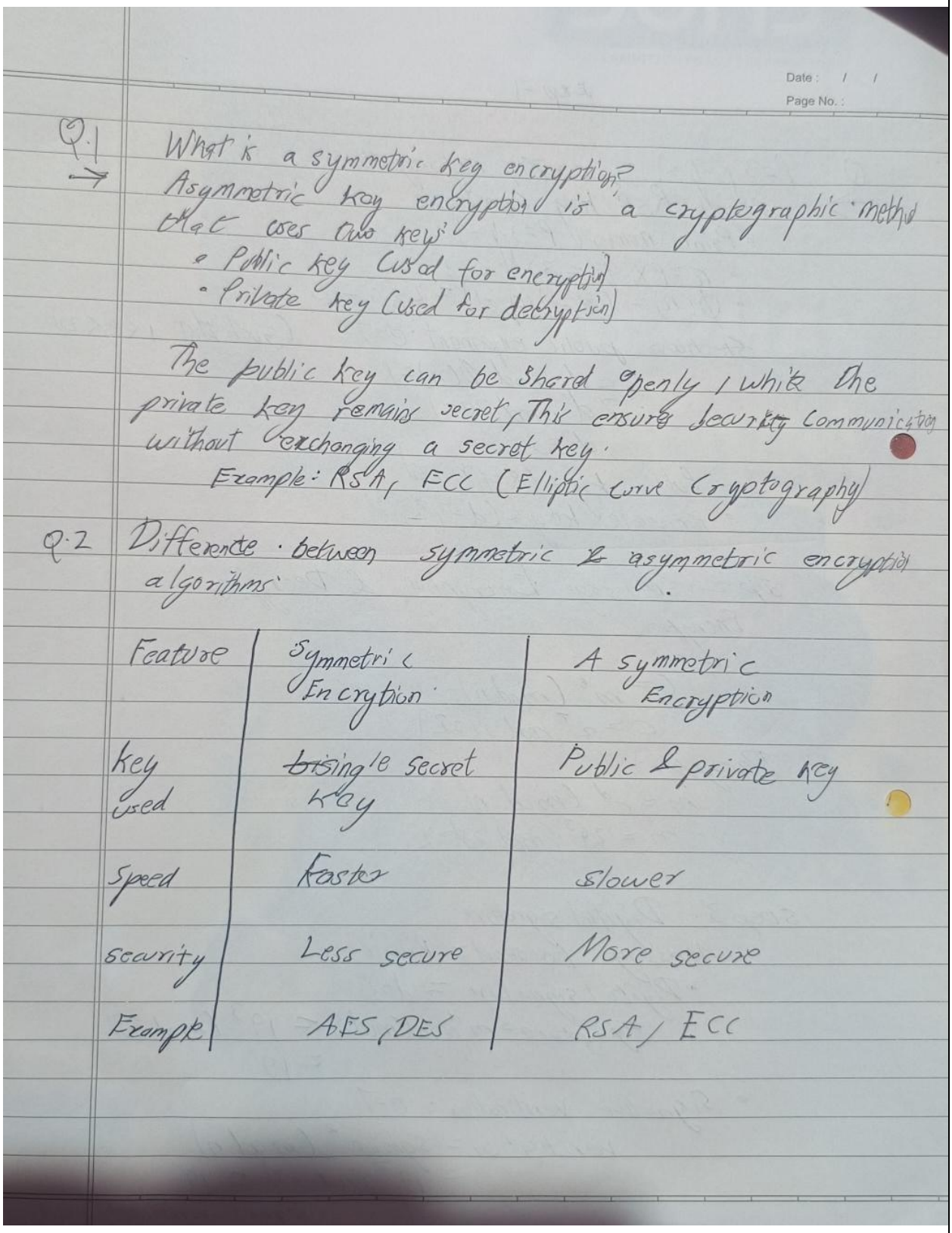
```
import hashlib

# Function to compute the Extended Euclidean Algorithm
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    g, x1, y1 = extended_gcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return g, x, y

# Function to compute modular inverse
def mod_inverse(a, m):
    g, x, _ = extended_gcd(a, m)
    if g != 1:
        raise Exception("Modular inverse does not exist")
    return x % m

# Function to check if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# RSA Key Generation
def generate_rsa_keys():
    while True:
        try:
            p = int(input("Enter a prime number p: "))
            q = int(input("Enter a prime number q: "))

            if not is_prime(p) or not is_prime(q):
                print("Both p and q should be prime numbers. Try again.")
            elif p == q:
                print("p and q should be distinct. Try again.")
            else:
                break
        except ValueError:
            print("Invalid input. Enter integers only.")
```

```
    n = p * q  # Compute n
    phi_n = (p - 1) * (q - 1)  # Compute φ(n)

    # Choose public exponent e
    while True:
        e = int(input(f"Enter public exponent e (1 < e < {phi_n} and gcd(e, {phi_n}) = 1): "))
        if 1 < e < phi_n and extended_gcd(e, phi_n)[0] == 1:
            break
        else:
            print(f"Invalid e. It must satisfy 1 < e < {phi_n} and gcd(e, {phi_n}) = 1.")

    # Compute private key exponent d
    d = mod_inverse(e, phi_n)

    print("\nPublic Key (e, n):", (e, n))
    print("Private Key (d, n):", (d, n))

    return (e, n), (d, n)

# Encrypt a message using the public key
def encrypt_message(message, public_key):
    e, n = public_key
    return pow(message, e, n)  # c = m^e % n

# Decrypt a message using the private key
def decrypt_message(ciphertext, private_key):
    d, n = private_key
    return pow(ciphertext, d, n)  # m = c^d % n

# Function to create a digital signature
def sign_message(message, private_key):
    hashed_msg = int(hashlib.sha256(message.encode()).hexdigest(), 16) % private_key[1]
# MODULO n
    return pow(hashed_msg, private_key[0], private_key[1])

# Function to verify a digital signature
def verify_signature(message, signature, public_key):
    hashed_msg = int(hashlib.sha256(message.encode()).hexdigest(), 16) % public_key[1]
# MODULO n
    decrypted_hash = pow(signature, public_key[0], public_key[1])
    return hashed_msg == decrypted_hash

# Main RSA Example with Digital Signature
def rsa_example():
    print("\n--- RSA Key Generation ---")
    public_key, private_key = generate_rsa_keys()
```

```
    print("\n--- Message Encryption & Decryption ---")
    while True:
        try:
            m = int(input("Enter a message to encrypt (integer less than n): "))
            if m >= public_key[1]:
                print(f"Message must be less than n ({public_key[1]}). Try again.")
            else:
                break
        except ValueError:
            print("Invalid input. Enter an integer.")

    ciphertext = encrypt_message(m, public_key)
    decrypted_message = decrypt_message(ciphertext, private_key)

    print("Encrypted Message (Ciphertext):", ciphertext)
    print("Decrypted Message:", decrypted_message)

    print("\n--- Digital Signature ---")
    message = input("Enter a message to sign: ")
    signature = sign_message(message, private_key)

    print("Digital Signature:", signature)

    verification_status = verify_signature(message, signature, public_key)
    print("Signature Verification:", "Valid" if verification_status else "Invalid")

# Run the RSA example
rsa_example()
```

B. Question of Curiosity: *(At least 3 questions should be handwritten)*

3 1. What is asymmetric key encryption?



Date : / /
Page No. :

Q.1 What is a symmetric key encryption?

→ Asymmetric key encryption is a cryptographic method that uses two keys:

- Public key (used for encryption)
- Private key (used for decryption)

The public key can be shared openly, while the private key remains secret. This ensures secure communication without exchanging a secret key.

Example: RSA, ECC (Elliptic curve Cryptography)

Q.2 Difference between symmetric & asymmetric encryption algorithms.

Feature	Symmetric Encryption	Asymmetric Encryption
Key used	Single secret key	Public & private key
Speed	Faster	Slower
Security	Less secure	More secure
Example	AES, DES	RSA, ECC

2. Explain different uses of RSA algorithm.

The RSA algorithm is widely used in various fields due to its strong encryption and digital signature capabilities. Here are some of the primary uses of RSA:

1. Secure Data Encryption

- RSA is used to encrypt sensitive data, ensuring that only the intended recipient can decrypt and access the information.
- It is often used in conjunction with symmetric key encryption (like AES) in hybrid encryption schemes. RSA encrypts the symmetric key, which is then used to encrypt the actual data.

2. Digital Signatures

- RSA allows the creation of digital signatures, which provide authentication and integrity verification for digital messages or documents.
- In this process, a sender signs a message with their private key. The recipient can verify the signature with the sender's public key, confirming the sender's identity and that the message has not been altered.

3. Secure Key Exchange

- RSA can be used to securely exchange keys over an insecure channel. The public key is used to encrypt the key, which can then be decrypted with the corresponding private key.
- This is typically seen in protocols like SSL/TLS, where RSA is used to exchange session keys securely.

4. SSL/TLS Protocols for Secure Web Communication

- RSA plays a crucial role in securing communication over the internet, particularly in HTTPS. During the SSL/TLS handshake, RSA is often used for key exchange and authentication, ensuring that communication between the client and server is private and authenticated.

5. Email Security (PGP and S/MIME)

- RSA is used in email encryption standards like PGP (Pretty Good Privacy) and S/MIME (Secure/Multipurpose Internet Mail Extensions) to ensure the confidentiality and authenticity of email communications.
- These systems use RSA to sign emails and encrypt email content, preventing unauthorized access and ensuring message integrity.

6. Blockchain and Cryptocurrency

- RSA is used in some blockchain and cryptocurrency systems to facilitate secure transactions, especially for private key management and signing of transaction data.
- Although blockchain often uses elliptic curve cryptography (ECC), RSA is still widely used in some implementations.

7. Authentication Systems

- RSA is used in authentication systems to verify users' identities. For example, it's used in two-factor authentication (2FA) systems, where RSA key pairs may be used to verify the identity of users trying to access secure systems.

8. Digital Certificates

- RSA is used in generating and verifying digital certificates. These certificates are essential in public key infrastructures (PKI) to confirm the authenticity of entities involved in a communication, such as a website's server.
- RSA enables the creation of the private and public keys associated with a digital certificate, which helps establish trust between parties.

9. Secure Online Transactions

- RSA is frequently used in securing online financial transactions, such as credit card transactions and e-commerce platforms, ensuring that the data sent between the user and merchant remains private and protected.

10. Virtual Private Networks (VPNs)

- RSA is used to establish secure connections in VPNs by enabling secure key exchanges during the setup of encrypted communication channels.

RSA remains one of the most trusted cryptographic algorithms because of its ability to provide secure communication, data integrity, and user authentication.

3. List and explain different possible attacks on RSA.

RSA is a robust encryption system, but it is not immune to attacks. Below are some common types of attacks that can potentially target RSA:

1. Brute Force Attack

- **Explanation**: In a brute force attack, the attacker attempts every possible key combination until they find the correct one. In the case of RSA, this would mean trying all possible values for pp, qq, ee, and dd to break the encryption.
- **Countermeasure**: Using large key sizes (e.g., 2048 or 4096-bit) makes brute-force attacks infeasible due to the time required to try all combinations.

2. Factoring Attack

- **Explanation**: The security of RSA is based on the difficulty of factoring large composite numbers into their prime factors. If an attacker can factor the modulus nn (the product of two prime numbers pp and qq), they can compute $\phi(n)$\phi(n) and then find the private key dd.
- **Countermeasure**: Using very large prime numbers for pp and qq (e.g., hundreds of digits) makes the factoring problem extremely difficult and time-consuming.

3. Chosen Ciphertext Attack (CCA)

- **Explanation**: In this attack, the attacker chooses a ciphertext, gets it decrypted (often through a side channel), and tries to deduce the private key or plaintext. A common variant is the **Adaptive Chosen Ciphertext Attack (CCA2)**, which can be very effective if the attacker can manipulate the ciphertext and receive responses from the decryption oracle.
- **Countermeasure**: RSA with padding schemes like **OAEP (Optimal Asymmetric Encryption Padding)** mitigates chosen ciphertext attacks by introducing randomness and complexity to the encryption process.

4. Chosen Plaintext Attack (CPA)

- **Explanation**: In a chosen plaintext attack, the attacker can choose a plaintext and obtain the corresponding ciphertext. By comparing multiple plaintext-ciphertext pairs, they may be able to deduce information about the private key or the plaintext.
- **Countermeasure**: Padding schemes (such as PKCS#1) help protect against chosen plaintext attacks by making it harder to derive patterns from the ciphertext.

5. Replay Attack

- **Explanation**: In this attack, the attacker intercepts and replays a previously transmitted ciphertext. Since RSA is a deterministic encryption algorithm, the same plaintext will always produce the same ciphertext. This allows an attacker to resend a valid message to trick the system into accepting it again.
- **Countermeasure**: Introducing **nonces** (random numbers used only once) or **timestamps** into the message ensures that each ciphertext is unique, preventing replay attacks.

6. Timing Attack

- **Explanation**: Timing attacks exploit the time variations in the computation of encryption or decryption. For instance, RSA decryption involves modular exponentiation, which may take slightly different amounts of time based on the values of the ciphertext. An attacker can measure these time variations to deduce information about the private key.
- **Countermeasure**: Implementing constant-time algorithms for decryption (e.g., using blinding techniques) helps avoid leaks through timing differences.

7. Side-Channel Attack

- **Explanation**: Side-channel attacks exploit physical information leaked during the RSA computation, such as power consumption, electromagnetic radiation, or CPU cache access patterns. By analyzing this information, an attacker can potentially recover the private key.
- **Countermeasure**: Techniques like **blinding** (randomizing the input to the RSA algorithm) and **masking** (hiding intermediate values) can be used to mitigate side-channel attacks.

8. Weak Key Attack

- **Explanation**: If the RSA key generation process is weak or flawed, such as when poor or predictable random numbers are used to select the primes pp and qq, the system might be vulnerable to attacks. For example, if pp and qq are too small or have some predictable patterns, an attacker could potentially guess them.
- **Countermeasure**: Ensuring that the random number generator used for selecting primes is cryptographically secure and producing large, unpredictable primes prevents weak key attacks.

9. Low Exponent Attack

- **Explanation**: If the public exponent ee is too small, for example, e=3e = 3, an attacker could exploit the mathematical properties of RSA to recover the plaintext from the ciphertext using a low-exponent attack. This is possible if the ciphertext is small enough relative to the modulus nn.
- **Countermeasure**: Using a larger public exponent (e.g., e=65537e = 65537) helps protect against this attack, as this value is large enough to make it difficult for an attacker to exploit the low exponent.

10. Social Engineering Attacks

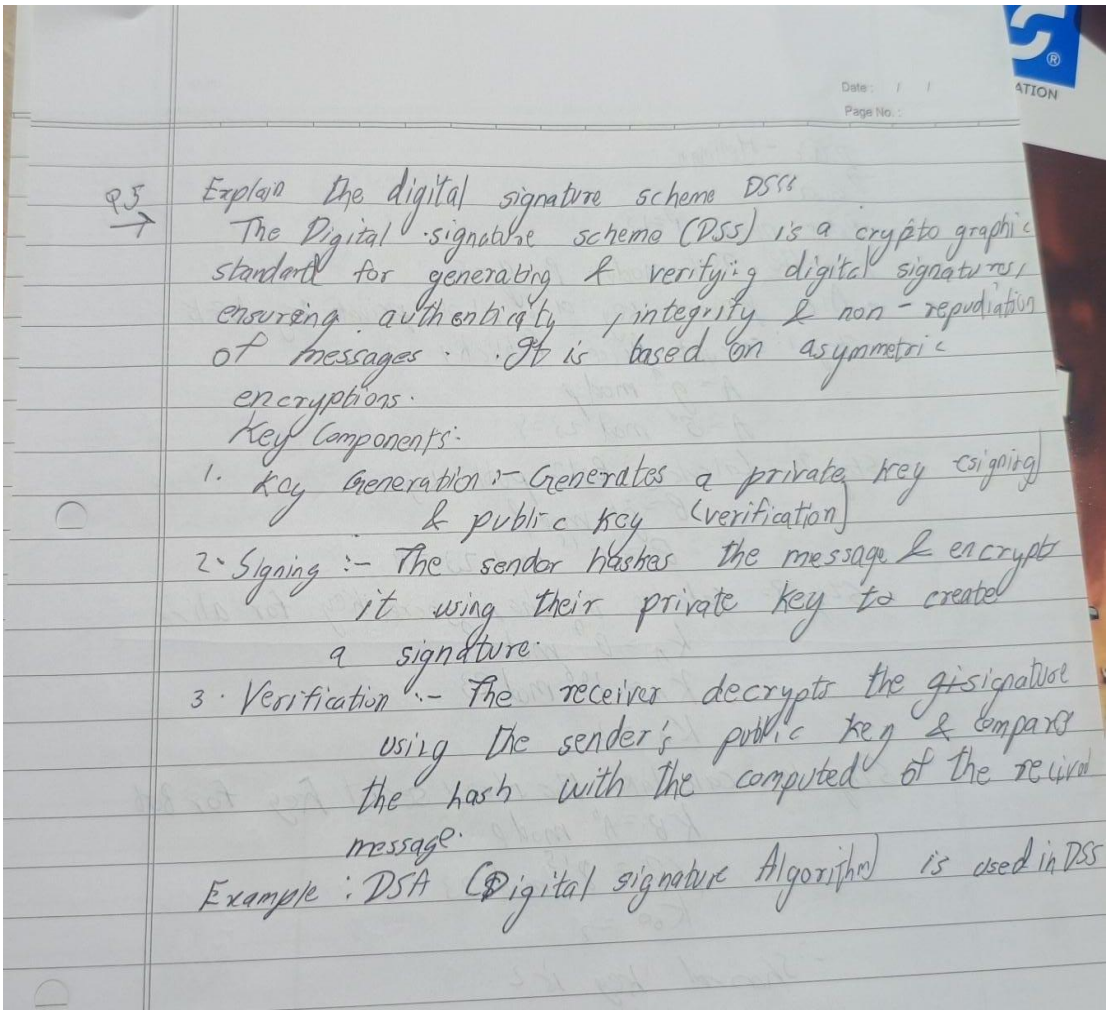
- **Explanation**: RSA keys can be compromised if the private key is not securely stored. Social engineering attacks can trick individuals into revealing their private keys, passphrases, or other sensitive information related to the key management process.
- **Countermeasure**: Employing strong authentication, securely storing private keys (e.g.,

using hardware security modules or secure key management systems), and educating users about security practices can mitigate social engineering risks.

11. Quantum Computing Attack (Future Threat)

- **Explanation**: Quantum computers, when fully realized, could break RSA encryption through Shor's algorithm, which can factor large numbers exponentially faster than classical algorithms. This would render RSA insecure in the face of quantum computing.
- **Countermeasure**: Research into **post-quantum cryptography** is ongoing, and new encryption methods that are resistant to quantum attacks are being developed to replace RSA in the future.

4. Explain the digital signature scheme DSS.



5. What are the limitations of using RSA for digital signatures? Answer in detail.

Here's a more concise version of the limitations of using RSA for digital signatures:

1. Key Size and Performance
    - RSA's security depends on key size; larger keys slow down operations. Performance can degrade in large-scale applications.
2. Quantum Computing
    - RSA is vulnerable to quantum computers, which can efficiently factor large numbers, breaking RSA's security.
3. Signature Size
    - RSA produces large signatures, which can be inefficient for systems with limited storage or bandwidth.
4. Poor Key Management
    - If the private key is exposed or mishandled, the signature system is compromised. Secure key storage is crucial.
5. Lack of Forward Secrecy
    - RSA doesn't provide forward secrecy; compromising the private key allows attackers to forge past signatures.
6. Weak Padding Schemes
    - Insecure padding schemes (e.g., PKCS#1 v1.5) can be vulnerable to attacks like Bleichenbacher's Attack.
7. Single Point of Failure
    - If the private key is lost or stolen, the signature system is rendered useless.
8. Scalability Issues
    - RSA's computational cost grows with key size and document size, which makes it less scalable for high-volume systems.

9. Vulnerabilities in Key Generation

- Weak or predictable primes in key generation can make the RSA keys easier to break.

10. Verification Vulnerabilities

- Poor implementation of signature verification could allow attackers to forge signatures or tamper with verification.

In summary, RSA faces limitations around performance, key management, and scalability, and it is vulnerable to quantum attacks. Alternatives like **ECDSA** and **post-quantum cryptography** are being considered for better security and efficiency.

B. Conclusion:

4

The experiment with the RSA algorithm and Digital Signature scheme highlighted key concepts in public-key cryptography. By understanding how to generate public and private keys, encrypt and decrypt messages, and create digital signatures, I learned how RSA ensures confidentiality, authenticity, and message integrity. It also demonstrated the importance of secure key management and the use of digital signatures for verifying the authenticity of messages. Through this, I gained a deeper understanding of how RSA is used in real-world applications, while also recognizing its limitations, such as performance issues and vulnerability to quantum attacks.