Comsats University Islamabad

(Attock Campus)

Lab Terminal

Student Name: Eman Ejaz

Registration No:(Sp24- BSE-052)

Subject: Information Security lab

Submitted to: Mam Ambreen Gul

Submission Date: 16th - December- 2025

Q1. Scenario: You are designing a secure email communication system that ensures confidentiality, integrity, and authenticity of messages. The system uses RSA for encryption and the Eelgamal for signing messages.

Tasks:

1. **Key Generation:**
2. **Encryption/Decryption using RSA :**
3. **Digital Signature and Verification using Elgamal**

(35 Marks)

[CLO 5 : Implement a cryptographic algorithm to ensure information security .

1.Key Generation:

Code:

```
# -------- RSA KEY GENERATION --------

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
    return None

def rsa_key_generation():
    p = 11
    q = 13
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 7
    d = mod_inverse(e, phi)
    return (e, n), (d, n)

# Generate keys
```

```
public_key, private_key = rsa_key_generation()

print("RSA Public Key:", public_key)
print("RSA Private Key:", private_key)
```

Output:

```
PS C:\Users\pak\Desktop\Python programs> & C:/Pr
programs/ISLabterminal.py"
RSA Public Key: (7, 143)
RSA Private Key: (103, 143)
PS C:\Users\pak\Desktop\Python programs>
```

2. Encryption/Decryption using RSA :

Code:

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
    raise Exception("Modular inverse not found")

def rsa_key_generation():
    p = 11
    q = 13
    n = p * q
    phi = (p - 1) * (q - 1)

    e = 7
    if gcd(e, phi) != 1:
        raise Exception("e and phi are not coprime")

    d = mod_inverse(e, phi)
    return (e, n), (d, n)

def rsa_encrypt_decrypt(message):
```

```
    # Key Generation
    public_key, private_key = rsa_key_generation()
    e, n = public_key
    d, _ = private_key

    # Encryption
    ciphertext = pow(message, e, n)

    # Decryption
    decrypted_message = pow(ciphertext, d, n)

    return ciphertext, decrypted_message


# -------- EXECUTION --------
message = 9
cipher, decrypted = rsa_encrypt_decrypt(message)

print("Original Message:", message)
print("Encrypted Message:", cipher)
print("Decrypted Message:", decrypted)
```

Output:

```
PS C:\Users\pak\Desktop\Python programs> & C:/Program F:
programs/ISLabterminal.py"
Original Message: 9
Encrypted Message: 48
Decrypted Message: 9
PS C:\Users\pak\Desktop\Python programs>
```

3: Digital Signature and Verification using Elgamal:

Code:

```
def elgamal_key_generation():
    p = 23       # Prime number
    g = 5        # Generator
    x = 6        # Private key
    y = pow(g, x, p)  # Public key
    return p, g, x, y

def elgamal_sign(p, g, x, message_hash):
    k = 7
    r = pow(g, k, p)
    k_inverse = pow(k, -1, p - 1)
    s = (k_inverse * (message_hash - x * r)) % (p - 1)
    return r, s
```

```python
def elgamal_verify(p, g, y, message_hash, r, s):
    left = pow(g, message_hash, p)
    right = (pow(y, r, p) * pow(r, s, p)) % p
    return left == right

# Key Generation
p, g, x, y = elgamal_key_generation()

print("\nElGamal Public Key (p, g, y):", (p, g, y))
print("ElGamal Private Key:", x)

# Signing
message_hash = 10
r, s = elgamal_sign(p, g, x, message_hash)
print("Digital Signature (r, s):", (r, s))

# Verification
is_valid = elgamal_verify(p, g, y, message_hash, r, s)
print("Signature Valid:", is_valid)
```

Output:

```
programs/ISLabterminal.py"

ElGamal Public Key (p, g, y): (23, 5, 8)
ElGamal Private Key: 6
Digital Signature (r, s): (17, 12)
Signature Valid: True
PS C:\Users\pak\Desktop\Python programs>
```

Question 2:

Project Questions:

1. **Justify your security method: Why is it suitable or better than other possible methods for your type of project?**

Answer:

Security Method Used:

Our project uses blockchain with Proof-of-Work (PoW) and SHA-256 hashing to secure transactions and blocks.

Why it is suitable or better:

- **Immutability:** Each block contains the hash of the previous block. Changing one block invalidates all subsequent blocks, making tampering evident.
- **Data integrity:** SHA-256 ensures that even a tiny change in a transaction or block produces a completely different hash.
- **Proof-of-Work:** Mining requires computational effort. This prevents easy modification of the chain, as recalculating PoW for all subsequent blocks is expensive.
- **Transparency & auditability:** All transactions are stored on-chain. Anyone with access can verify the chain's validity using the is_valid() method.

Comparison with other methods:

- Compared to traditional databases**,** blockchain is more resistant to tampering without needing a central authority.
- Compared to simple encryption**,** blockchain not only protects confidentiality (if needed) but also ensures integrity, order, and traceability of all transactions.
- For a project demonstrating **secure** transaction storage and tamper detection**,** blockchain is more suitable than alternatives like symmetric encryption or simple hashing alone.

2. Identify one possible vulnerability or weakness in your current system. How could an attacker misuse it?

Answer:

Vulnerability:

- Our current system does not implement network-level security, user authentication, or encryption for transaction data**.**
- Transactions are stored in plain text (sender, receiver, amount), which means anyone with access to the system files or memory could read all transaction details.
- There is **no** verification of user identity before adding transactions, so anyone who can run the code could submit transactions on behalf of someone else.
- The system is single-node and local**,** so while tampering is detectable within the chain, there is no protection against malicious nodes in a networked environment.

How an attacker could misuse it:

- **Manual manipulation of pending transactions:** An attacker could add fake or fraudulent transactions before the block is mined, potentially giving themselves or another user undeserved funds.
- **Eavesdropping:** If the system were networked, an attacker could intercept transaction data because it is not encrypted, gaining sensitive information about users and their balances.

- **Tampering with mined blocks:** While mining ensures PoW, an attacker could attempt to tamper with blocks if they have sufficient computing power. In a distributed setting, a 51% attack—where the attacker controls the majority of mining power,could allow them to re-mine blocks, double-spend, or reorder transactions.
- **Unauthorized access:** Without authentication, anyone with access to the code or system can exploit it, undermining the trustworthiness of the blockchain.

3. Suggest one realistic improvement to enhance the security of your project. Briefly explain how it would work.

Answer:

Improvement:

- Implement digital signatures for transactions using asymmetric cryptography (RSA or ECC).

How it would work:

1. Each user generates a **public/private key pair**. The private key is kept secret, while the public key is shared with the network.
2. When creating a transaction, the sender signs the transaction data (sender, receiver, amount, timestamp) with their private key, creating a digital signature**.**
3. The system verifies the transaction by checking the signature against the sender's public key before adding it to the pending transactions list. Only transactions with valid signatures are accepted.
4. Once verified, the transaction is included in a block and mined as usual. Any tampering with the transaction data after signing will cause the signature verification to fail, ensuring integrity.

Benefits:

- **Authenticity:** Only the owner of the private key can create a valid transaction. This ensures that transactions cannot be forged.
- **Integrity:** Any change in the transaction data invalidates the signature, making tampering detectable even before mining.
- **Non-repudiation:** The sender cannot deny sending the transaction, as the signature uniquely identifies them.
- **Enhanced security:** Even if an attacker gains access to the pending transaction pool, they cannot inject fraudulent transactions without the sender's private key.
- **Compatibility with blockchain:** Digital signatures complement the existing hashing and Proof-of-Work system, reinforcing the tamper-resistance of both transaction content and block structure.