Degree of BSc (Hons) in Information Technology

Faculty of Information Technology

Horizon Campus

IT32023 - Information Assurance and Network Security Intake11 Assignment01

ITBIN-2211-0330

R.K.R Jayathissa

# Question 1

1. Part A

```python
import string

class CipherAssignment:
    def __init__(self, full_name):
        # Store the full name
        self.full_name = full_name

        # Preprocess the name: remove spaces and convert to uppercase for consistency
        self.plaintext = full_name.replace(" ", "").upper()

        # Standard alphabet (A-Z)
        self.alphabet = string.ascii_uppercase

        # The specific key provided in the assignment for Kamasutra Cipher
        self.kamasutra_key = "GJMQTVZADIORUBCEFHKLNPSWXY"

    def kamasutra_encrypt(self):
        """
        Encrypts the stored plaintext using the Kamasutra substitution cipher.
        Mapping is done based on the index of the character in the standard alphabet.
        """
        ciphertext = []

        for char in self.plaintext:
            if char in self.alphabet:
                # Find the index of the character in the standard alphabet (0-25)
                index = self.alphabet.index(char)

                # Find the corresponding character in the key
                encrypted_char = self.kamasutra_key[index]
                ciphertext.append(encrypted_char)
            else:
                # Keep non-alphabetic characters unchanged (if any)
                ciphertext.append(char)

        # Join the list of characters to form the final string
        return "".join(ciphertext)

# --- Main Execution ---

# 1. Input your full name here
# REPLACE "YOUR FULL NAME HERE" with your actual name (e.g., "KASUN PERERA")
your_full_name = "Kavishka Rasanjana Jyathissa"

# 2. Create an object of the class
cipher_tool = CipherAssignment(your_full_name)

# 3. Perform Kamasutra Encryption (Question 01 - Part a)
encrypted_name = cipher_tool.kamasutra_encrypt()

# 4. Display the results
print(f"Original Name: {cipher_tool.full_name}")
print(f"Cleaned Plaintext: {cipher_tool.plaintext}")
print(f"Kamasutra Ciphertext: {encrypted_name}")
```

```
Original Name: Kavishka Rasanjana Jyathissa
Cleaned Plaintext: KAVISHKARASANJANAJYATHISSA
Kamasutra Ciphertext: OGPDKAOGHGKGBIGBGIXGLADKKG
```

Part B

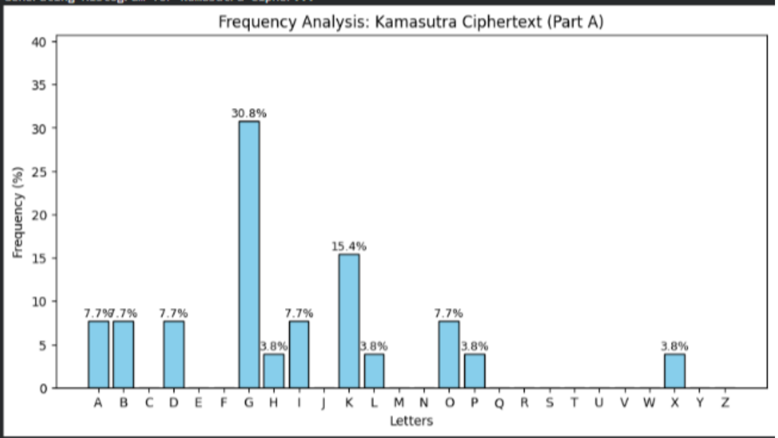
```python
# --- Part (b): Modular Function Cipher Execution ---
modular_result = cipher_tool.modular_encrypt()

print(f"Part (b) Output (Modular Cipher): {modular_result}")
```

```
Part (b) Output (Modular Cipher): YGXKCDYGVGCGTRGTGRGSJDKCCG
```

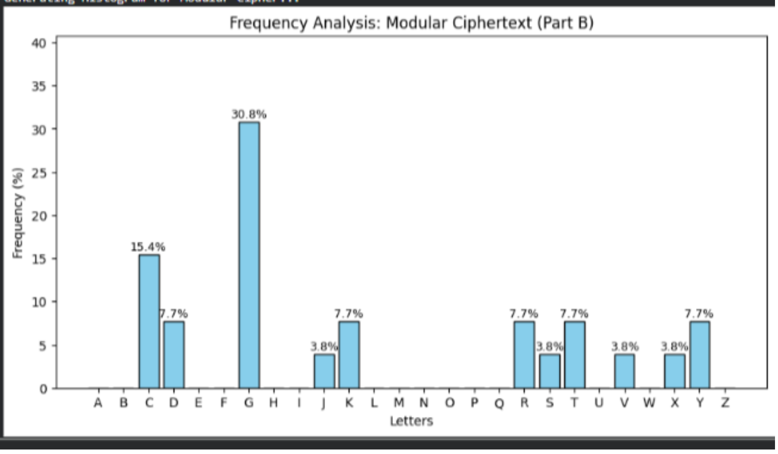
Part C

```python
# --- Part (c): Frequency Analysis & Histograms ---

print("Generating Histogram for Kamasutra Cipher...")
cipher_tool.plot_histogram(kamasutra_result, "Kamasutra Ciphertext (Part A)")

print("\nGenerating Histogram for Modular Cipher...")
cipher_tool.plot_histogram(modular_result, "Modular Ciphertext (Part B)")
```

```
Generating Histogram for Kamasutra Cipher...
```




```
Generating Histogram for Modular Cipher...
```



## Question 2

1.

```
    # If one point is None (Infinity), return the other
    if P is None: return Q
    if Q is None: return P

    x1, y1 = P
    x2, y2 = Q

    # Check if points are vertical reflections (sum is Infinity)
    if x1 == x2 and (y1 != y2 or y1 == 0):
        return None

    if x1 == x2 and y1 == y2:
        # Point Doubling Case: slope m = (3x^2 + a) / 2y
        # We use modular inverse for division
        numerator = (3 * x1**2 + self.a)
        denominator = pow(2 * y1, -1, self.p)
        m = (numerator * denominator) % self.p
    else:
        # Point Addition Case: slope m = (y2 - y1) / (x2 - x1)
        numerator = (y2 - y1)
        denominator = pow(x2 - x1, -1, self.p)
        m = (numerator * denominator) % self.p

    # Calculate new point coordinates
    x3 = (m**2 - x1 - x2) % self.p
    y3 = (m * (x1 - x3) - y1) % self.p

    return (x3, y3)

def scalar_mult(self, k, Point):
    """
    Performs Scalar Multiplication: k * P
    This is used to generate Public Keys and Shared Secrets.
    Uses the 'Double-and-Add' algorithm efficiently.
    """
    result = None # Start at Infinity
    addend = Point

    while k > 0:
        # If the current bit is 1, add the current point
        if k % 2 == 1:
            result = self.point_add(result, addend)

        # Double the point for the next bit
        addend = self.point_add(addend, addend)
        k //= 2 # Shift bits to the right

    return result
```

2.

```python
# --- Configuration ---

# Step (a): Define Curve Parameters
# Equation: y^2 = x^3 + 2x + 2 (mod 17)
a = 2
b = 2
p = 17

# Create the ECC Object
ecc = ECCKeyExchange(a, b, p)

# Step (b): Choose a Base Point G
# G must satisfy the curve equation
G = (5, 1)

print("--- ECC System Setup ---")
print(f"Curve Equation: y^2 = x^3 + {a}x + {b} (mod {p})")
print(f"Base Point G: {G}")
```

```
--- ECC System Setup ---
Curve Equation: y^2 = x^3 + 2x + 2 (mod 17)
Base Point G: (5, 1)
```

3.

```python
import random

# --- Key Generation ---

# Step (c): Tim's Keys
# Tim selects a random private key (must be less than p)
tim_private_key = 3   # Example small number
tim_public_key = ecc.scalar_mult(tim_private_key, G)

print("\n--- Tim's Keys ---")
print(f"Tim's Private Key: {tim_private_key}")
print(f"Tim's Public Key:  {tim_public_key}")


# Step (d): Stephen's Keys
# Stephen selects a random private key
stephen_private_key = 10 # Example small number
stephen_public_key = ecc.scalar_mult(stephen_private_key, G)

print("\n--- Stephen's Keys ---")
print(f"Stephen's Private Key: {stephen_private_key}")
print(f"Stephen's Public Key:  {stephen_public_key}")
```

```
--- Tim's Keys ---
Tim's Private Key: 3
Tim's Public Key:  (10, 6)

--- Stephen's Keys ---
Stephen's Private Key: 10
Stephen's Public Key:  (7, 11)
```

4.

```python
# --- Shared Secret Computation ---

# Step (e): Calculate Shared Secrets
# Tim computes: tim_private * stephen_public
tim_shared_secret = ecc.scalar_mult(tim_private_key, stephen_public_key)

# Stephen computes: stephen_private * tim_public
stephen_shared_secret = ecc.scalar_mult(stephen_private_key, tim_public_key)


# Step (f): Display and Verify
print("\n--- Verification ---")
print(f"Tim's Calculated Shared Secret:     {tim_shared_secret}")
print(f"Stephen's Calculated Shared Secret: {stephen_shared_secret}")

if tim_shared_secret == stephen_shared_secret:
    print("\nSUCCESS: Keys match! Secure exchange successful.")
else:
    print("\nERROR: Keys do not match.")
```

```
--- Verification ---
Tim's Calculated Shared Secret:     (13, 10)
Stephen's Calculated Shared Secret: (13, 10)

SUCCESS: Keys match! Secure exchange successful.
```