3*3 Hill Cipher Encryption and Decryption

In [4]:

```python
import numpy as np

def encrypt(text, key_matrix):
    # Convert text to uppercase and remove spaces
    text = text.replace(" ", "").upper()

    # Pad the text with 'X' if its length is not a multiple of 3
    while len(text) % 3 != 0:
        text += "X"

    # Initialize the result
    encrypted_text = ""

    # Loop through the text in blocks of 3 characters
    for i in range(0, len(text), 3):
        block = text[i:i+3]

        # Convert the block to a vector
        block_vector = np.array([ord(char) - ord('A') for char in block])

        # Perform matrix multiplication
        result_vector = np.dot(key_matrix, block_vector) % 26

        # Convert the result vector back to characters
        encrypted_block = "".join([chr(result + ord('A')) for result in r

        encrypted_text += encrypted_block
    return encrypted_text

def decrypt(encrypted_text, key_matrix):
    # Calculate the modular inverse of the determinant of the key matrix
    determinant = int(np.round(np.linalg.det(key_matrix)))
    determinant_inverse = None

    for i in range(26):
        if (i * determinant) % 26 == 1:
            determinant_inverse = i
            break

    if determinant_inverse is None:
        raise ValueError("The determinant has no modular inverse")

    # Calculate the adjugate of the key matrix
    key_matrix_inverse = np.round(np.linalg.inv(key_matrix) * determinant

    # Initialize the result
    decrypted_text = ""

    # Loop through the encrypted text in blocks of 3 characters
    for i in range(0, len(encrypted_text), 3):
        block = encrypted_text[i:i+3]

        # Convert the block to a vector
        block_vector = np.array([ord(char) - ord('A') for char in block])

        # Perform matrix multiplication with the inverse key matrix
        result_vector = np.dot(key_matrix_inverse, block_vector) % 26
```

```python

        # Convert the result vector back to characters
        decrypted_block = "".join([chr(int(result) + ord('A')) for result

        decrypted_text += decrypted_block

    return decrypted_text

# key matrix
key_matrix = np.array([[6, 24, 1], [13, 16, 10], [20, 17, 15]])

# plaintext
plaintext = input("Enter the plaintext :- ")
#plaintext = "Amar Deep"

# Encrypt the plaintext
encrypted_text = encrypt(plaintext, key_matrix)
print("Encrypted:", encrypted_text)

# Decrypt the encrypted text
decrypted_text = decrypt(encrypted_text, key_matrix)
print("Decrypted:", decrypted_text)

```

```
Enter the plaintext :- ACT
Encrypted: POH
Decrypted: AYH
```

2*2 Hill Cipher Encryption and Decryption

In [15]:

```
import numpy as np

def encrypt(text, key_matrix):
    # Convert text to uppercase and remove spaces
    text = text.replace(" ", "").upper()

    # Pad the text with 'X' if its length is not even
    if len(text) % 2 != 0:
        text += "X"

    # Initialize the result
    encrypted_text = ""

    # Loop through the text in blocks of 2 characters
    for i in range(0, len(text), 2):
        block = text[i:i+2]

        # Convert the block to a vector
        block_vector = np.array([ord(char) - ord('A') for char in block])

        # Perform matrix multiplication
        result_vector = np.dot(key_matrix, block_vector) % 26

        # Convert the result vector back to characters
        encrypted_block = "".join([chr(result + ord('A')) for result in r

        encrypted_text += encrypted_block

    return encrypted_text

def decrypt(encrypted_text, key_matrix):
    # Calculate the modular inverse of the determinant of the key matrix
    determinant = int(np.round(np.linalg.det(key_matrix)))
    determinant_inverse = None

    for i in range(26):
        if (i * determinant) % 26 == 1:
            determinant_inverse = i
            break

    if determinant_inverse is None:
        raise ValueError("The determinant has no modular inverse")

    # Calculate the adjugate of the key matrix
    key_matrix_inverse = np.round(np.linalg.inv(key_matrix) * determinant

    # Initialize the result
    decrypted_text = ""

    # Loop through the encrypted text in blocks of 2 characters
    for i in range(0, len(encrypted_text), 2):
        block = encrypted_text[i:i+2]

        # Convert the block to a vector
        block_vector = np.array([ord(char) - ord('A') for char in block])

        # Perform matrix multiplication with the inverse key matrix
```

```
        result_vector = np.dot(key_matrix_inverse, block_vector) % 26

        # Convert the result vector back to characters
        decrypted_block = "".join([chr(int(result) + ord('A')) for result

        decrypted_text += decrypted_block

    return decrypted_text

# Example key matrix
key_matrix = np.array([[5, 8], [17, 3]])

# Example plaintext
plaintext = input("Enter the plaintext :- ")

# Encrypt the plaintext
encrypted_text = encrypt(plaintext, key_matrix)
print("Encrypted:", encrypted_text)

# Decrypt the encrypted text
decrypted_text = decrypt(encrypted_text, key_matrix)
print("Decrypted:", decrypted_text)

```

```
Enter the plaintext :- AMR
Encrypted: SKJU
Decrypted: AEXZ
```