

3*3 Hill Cipher Encryption and Decryption

In [4]:

```

1  import numpy as np
2
3  def encrypt(text, key_matrix):
4      # Convert text to uppercase and remove spaces
5      text = text.replace(" ", "").upper()
6
7      # Pad the text with 'X' if its length is not a multiple of 3
8      while len(text) % 3 != 0:
9          text += "X"
10
11     # Initialize the result
12     encrypted_text = ""
13
14     # Loop through the text in blocks of 3 characters
15     for i in range(0, len(text), 3):
16         block = text[i:i+3]
17
18         # Convert the block to a vector
19         block_vector = np.array([ord(char) - ord('A') for char in block])
20
21         # Perform matrix multiplication
22         result_vector = np.dot(key_matrix, block_vector) % 26
23
24         # Convert the result vector back to characters
25         encrypted_block = "".join([chr(result + ord('A')) for result in result_vector])
26
27         encrypted_text += encrypted_block
28     return encrypted_text
29
30 def decrypt(encrypted_text, key_matrix):
31     # Calculate the modular inverse of the determinant of the key matrix
32     determinant = int(np.round(np.linalg.det(key_matrix)))
33     determinant_inverse = None
34
35     for i in range(26):
36         if (i * determinant) % 26 == 1:
37             determinant_inverse = i
38             break
39
40     if determinant_inverse is None:
41         raise ValueError("The determinant has no modular inverse")
42
43     # Calculate the adjugate of the key matrix
44     key_matrix_inverse = np.round(np.linalg.inv(key_matrix) * determinant)
45
46     # Initialize the result
47     decrypted_text = ""
48
49     # Loop through the encrypted text in blocks of 3 characters
50     for i in range(0, len(encrypted_text), 3):
51         block = encrypted_text[i:i+3]
52
53         # Convert the block to a vector
54         block_vector = np.array([ord(char) - ord('A') for char in block])
55
56         # Perform matrix multiplication with the inverse key matrix
57         result_vector = np.dot(key_matrix_inverse, block_vector) % 26

```

```
58
59     # Convert the result vector back to characters
60     decrypted_block = "".join([chr(int(result) + ord('A')) for result
61
62         decrypted_text += decrypted_block
63
64     return decrypted_text
65
66 # key matrix
67 key_matrix = np.array([[6, 24, 1], [13, 16, 10], [20, 17, 15]])
68
69 # plaintext
70 plaintext = input("Enter the plaintext :- ")
71 #plaintext = "Amar Deep"
72
73 # Encrypt the plaintext
74 encrypted_text = encrypt(plaintext, key_matrix)
75 print("Encrypted:", encrypted_text)
76
77 # Decrypt the encrypted text
78 decrypted_text = decrypt(encrypted_text, key_matrix)
79 print("Decrypted:", decrypted_text)
80
```

Enter the plaintext :- ACT

Encrypted: POH

Decrypted: AYH

2*2 Hill Cipher Encryption and Decryption

In [15]:

```

1  import numpy as np
2
3  def encrypt(text, key_matrix):
4      # Convert text to uppercase and remove spaces
5      text = text.replace(" ", "").upper()
6
7      # Pad the text with 'X' if its length is not even
8      if len(text) % 2 != 0:
9          text += "X"
10
11     # Initialize the result
12     encrypted_text = ""
13
14     # Loop through the text in blocks of 2 characters
15     for i in range(0, len(text), 2):
16         block = text[i:i+2]
17
18         # Convert the block to a vector
19         block_vector = np.array([ord(char) - ord('A') for char in block])
20
21         # Perform matrix multiplication
22         result_vector = np.dot(key_matrix, block_vector) % 26
23
24         # Convert the result vector back to characters
25         encrypted_block = "".join([chr(result + ord('A')) for result in result_vector])
26
27         encrypted_text += encrypted_block
28
29     return encrypted_text
30
31 def decrypt(encrypted_text, key_matrix):
32     # Calculate the modular inverse of the determinant of the key matrix
33     determinant = int(np.round(np.linalg.det(key_matrix)))
34     determinant_inverse = None
35
36     for i in range(26):
37         if (i * determinant) % 26 == 1:
38             determinant_inverse = i
39             break
40
41     if determinant_inverse is None:
42         raise ValueError("The determinant has no modular inverse")
43
44     # Calculate the adjugate of the key matrix
45     key_matrix_inverse = np.round(np.linalg.inv(key_matrix) * determinant)
46
47     # Initialize the result
48     decrypted_text = ""
49
50     # Loop through the encrypted text in blocks of 2 characters
51     for i in range(0, len(encrypted_text), 2):
52         block = encrypted_text[i:i+2]
53
54         # Convert the block to a vector
55         block_vector = np.array([ord(char) - ord('A') for char in block])
56
57         # Perform matrix multiplication with the inverse key matrix

```

```
58     result_vector = np.dot(key_matrix_inverse, block_vector) % 26
59
60     # Convert the result vector back to characters
61     decrypted_block = "".join([chr(int(result) + ord('A')) for result
62                                in result_vector])
63     decrypted_text += decrypted_block
64
65     return decrypted_text
66
67 # Example key matrix
68 key_matrix = np.array([[5, 8], [17, 3]])
69
70 # Example plaintext
71 plaintext = input("Enter the plaintext :- ")
72
73 # Encrypt the plaintext
74 encrypted_text = encrypt(plaintext, key_matrix)
75 print("Encrypted:", encrypted_text)
76
77 # Decrypt the encrypted text
78 decrypted_text = decrypt(encrypted_text, key_matrix)
79 print("Decrypted:", decrypted_text)
80
```

Enter the plaintext :- AMR

Encrypted: SKJU

Decrypted: AEXZ