25) Write a C program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext–ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted. Implement in C programming.

PROGRAM:-

```
import numpy as np
def modinv(a, m):
    """Modular inverse of a under modulo m"""
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    raise ValueError("Modular inverse does not exist")
def matrix_mod_inv(matrix, modulus):
    """Find the inverse of a matrix under modulus"""
    det = int(np.round(np.linalg.det(matrix))) % modulus
    det_inv = modinv(det, modulus)
    matrix_adj = np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    return (det_inv * matrix_adj) % modulus
def text_to_numbers(text):
    return [ord(char.upper()) - ord('A') for char in text if char.isalpha()]
def numbers_to_text(numbers):
    return ''.join([chr(num % 26 + ord('A')) for num in numbers])
def chunk_text(text, size):
    nums = text_to_numbers(text)
    while len(nums) % size != 0:
        nums.append(0)
    return [nums[i:i + size] for i in range(0, len(nums), size)]
def derive_key(plaintext, ciphertext, block_size):
    P_chunks = chunk_text(plaintext, block_size)
    C_chunks = chunk_text(ciphertext, block_size)
    if len(P_chunks) < block_size:
        raise ValueError("Not enough plaintext/ciphertext to form square matrix")
```

```
    P = np.array(P_chunks[:block_size]).T
    C = np.array(C_chunks[:block_size]).T
    try:
        P_inv = matrix_mod_inv(P, 26)
        K = (C @ P_inv) % 26
        return K.astype(int)
    except Exception as e:
        print("Error in key derivation:", e)
        return None
def encrypt(plaintext, key):
    block_size = key.shape[0]
    P_chunks = chunk_text(plaintext, block_size)
    ciphertext = ''
    for chunk in P_chunks:
        vec = np.array(chunk)
        enc = key.dot(vec) % 26
        ciphertext += numbers_to_text(enc)
    return ciphertext
# Example with block size 2
plaintext = "HELP"
ciphertext = "IZWX"
block_size = 2
key_matrix = derive_key(plaintext, ciphertext, block_size)
print("Recovered Key Matrix:\n", key_matrix)
if key_matrix is not None:
    new_plaintext = "OKAY"
    encrypted = encrypt(new_plaintext, key_matrix)
    print(f"Encrypted '{new_plaintext}' -> '{encrypted}'")
```

OUTPUT:-

```
Recovered Key Matrix:
 [[ 8 18]
 [ 3 23]]
Encrypted 'OKAY' -> 'GMQG'
```
