```
import numpy as np

def mod_inverse(a, m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    raise ValueError(f"No modular inverse for {a} under mod {m}")

def text_to_numbers(text):
    return [ord(c) - ord('a') for c in text.lower()]

def numbers_to_text(numbers):
    return ''.join([chr((num % 26) + ord('a')) for num in numbers])

def create_key_matrix(key, n):
    key_nums = text_to_numbers(key)
    if len(key_nums) != n * n:
        raise ValueError(f"Key must be {n*n} characters for {n}x{n} matrix.")
    return np.array(key_nums).reshape(n, n)

def matrix_mod_inverse(matrix, modulus):
    det = int(round(np.linalg.det(matrix)))  # determinant
    det_inv = mod_inverse(det, modulus)
    matrix_mod_inv = (det_inv * np.round(det * np.linalg.inv(matrix)).astype(int)) % modulus
    return matrix_mod_inv

def hill_cipher_encrypt(plaintext, key):
    plaintext = plaintext.replace(" ", "").lower()
    key_length = len(key)
    n = int(key_length ** 0.5)
    if n * n != key_length:
        raise ValueError("Key length must be a perfect square.")

    while len(plaintext) % n != 0:
        plaintext += 'x'  # Padding

    key_matrix = create_key_matrix(key, n)
    ciphertext = ""

    for i in range(0, len(plaintext), n):
        block = plaintext[i:i+n]
        block_vector = np.array(text_to_numbers(block))
        encrypted_vector = np.dot(key_matrix, block_vector) % 26
        ciphertext += numbers_to_text(encrypted_vector)

    return ciphertext, key_matrix

def hill_cipher_decrypt(ciphertext, key_matrix):
    n = key_matrix.shape[0]
    inverse_key_matrix = matrix_mod_inverse(key_matrix, 26)
    plaintext = ""
```

```
    for i in range(0, len(ciphertext), n):
        block = ciphertext[i:i+n]
        block_vector = np.array(text_to_numbers(block))
        decrypted_vector = np.dot(inverse_key_matrix, block_vector) % 26
        plaintext += numbers_to_text(decrypted_vector)

    return plaintext

# Main program
if __name__ == "__main__":
    plaintext = input("Enter the plaintext: ").strip()
    key = input("Enter the key (perfect square length, e.g., 4, 9, 16): ").strip()

    try:
        ciphertext, key_matrix = hill_cipher_encrypt(plaintext, key)
        print("
Encrypted ciphertext:", ciphertext)

        decrypted = hill_cipher_decrypt(ciphertext, key_matrix)
        print("Decrypted plaintext:", decrypted)
    except Exception as e:
        print("Error:", e)
```