3. WRITTEN RESPONSES

3 a.

3.a.i.

The program solves the problem of password security through encrypting/decrypting an input using a key. The program implements the Hill Cipher, a method which becomes increasingly difficult to encrypt by hand as the size of a key increases. The program offers security and practicality within cryptography.

3.a.ii.

The program prompts the user for a “Password” and then prompts for a “Key”. The inputs are then converted into unicode and matrices. A final input selects either the “Encryption” or “Decryption” function. The main functionality of the program is matrix multiplication. The resulting matrix of encrypted unicodes is then converted into an string output.

3.a.iii.

The “Password” inputs are “ACT” and “ATTACK”. The “Key” inputs are “GYBNQKURP” and “BCDEFGHIJ”. The selector input was “1” or encryption. The important output for the first scenario is “POH”. The important output for the second scenario is “Your key can not be inverted! Please restart the program.”

3 b.

3.b.i.

```
lengthMatrix = rounded(math.sqrt(len(key)))
key = [alphabet[letter] for letter in key]
key = [key[i:i+lengthMatrix] for i in range(0, len(key), lengthMatrix)]
```

3.b.ii.

```
    isValid = determinant(key)
    if isValid == 0 or isValid % 2 == 0 or isValid % 13 == 0:
        print('Your key can not be inverted! Please restart the program!')
        app.stop()
        return None
    encryptedPassword = ''
    index = 0
    for cluster in password:
        for i in range(len(key)):
            for j in range(len(key[0])):
                index += key[i][j] * cluster[j % len(cluster)]
            index %= 26
            encryptedPassword = encryptedPassword + chr(index + 65)
            index = 0
```

3.b.iii.

The list is called 'key'.

3.b.iv.

The data of the list represents the alphabetic index of an input's letters. The key is manipulated into a square matrix or nestled lists.

3.b.v.

While one could access the index of the key's letters without a list, it would require constant conversion of strings to unicode with the ord() function and subtraction of 65. You could store these converted unicodes into a string, but it would also require endless str() and int() conversion.

More importantly, determining the invertibility of a key would be very difficult using strings. Finding the determinant of a matrix requires Gaussian Elimination. Row swapping could be reasonable with clever string splicing, but scaling a row with an integer or subtracting/adding two rows would be immensely tedious or even impossible. It would require even more ord()/int() conversion and abnormal string concatenation. Modifying the specific value inside a matrix/list's rows and columns is a much easier job for the user and the memory of the computer.

3 c.

3.c.i.

```
def encryption(password, key):
    isValid = determinant(key)
    if isValid == 0 or isValid % 2 == 0 or isValid % 13 == 0:
        print('Your key can not be inverted! Please restart the program!')
        app.stop()
        return None
    encryptedPassword = ''
    index = 0
    for cluster in password:
        for i in range(len(key)):
            for j in range(len(key[0])):
                index += key[i][j] * cluster[j % len(cluster)]
            index %= 26
            encryptedPassword = encryptedPassword + chr(index + 65)
            index = 0
    print('Your encrypted text is ' + encryptedPassword + '.')
    return encryptedPassword
```

3.c.ii.

```
selector = app.getTextInput('Do you wish to encrypt or decrypt? (1 or 2)')
if selector == '1':
    encryption(password, key)
```

3.c.iii.

The identified procedure multiplies the several nx1 matrices of the password parameter by a key parameter of nxn. Using a handmade function or importing a linear algebra module, the procedure will also take the determinant of the key matrix determining if it has a modular inverse. The output prints the encrypted unicodes as a string. The procedure is the encryption functionality of the program as the Hill Cipher is essentially matrix multiplication.

3.c.iv.

Use a home-made function or import a linear algebra module to find the determinant of your key parameter. If the determinant is zero or a multiple of 2/13, halt the program.

The algorithm's goal is to multiply the rows of our key by the rows of our chunked password. A 'index' variable stores the products of each row-to-row multiplication.

The algorithm stores the encrypted letters with a string variable (' '). For each chunk of our password, the algorithm will loop over the key's rows and columns using two nestled for loops. The variable of each loop represents the index of the row/column the procedure accesses in the key. For every row(i) in the key, the algorithm traverses the columns (j) in the row accessing each item. After indexing the key for its i row and j column, it multiplies the value of this element by the value of the j-th item in your chunk. The algorithm adds that product to the index variable. After it has iterated/traveled over a row, the algorithm adds 65 to the index variable to create an unicode. The algorithm converts this sum into a string using the chr() function, concatenates it to the string variable, and resets the index variable to zero. The algorithm displays the encrypted string after multiplying every chunk.

3 d.

3.d.i.

First call:

encryption([0, 2 , 19], [[6, 24, 1], [13, 16, 10], [20, 17, 15]])

Second call:

encryption([0,19,19,0,2,10], [[2, 3,4], [4, 5,6], [7,8,9]])

3 d.ii.

Condition(s) tested by first call:

Is the determinant of [[6, 24, 1], [13, 16, 10], [20, 17, 15]] a non-zero value? Is the determinant of the matrix a multiple of 2 or 13?

Condition(s) tested by second call:

Is the determinant of
[[2, 3,4], [4, 5,6], [7,8,9]]
a non-zero value? Is the determinant of the matrix a multiple of 2 or 13?

3.d.iii.

Results of the first call:

The key matrix is invertible, and the function proceeds to the encryption procedure and outputs ‘POH’.

Results of the second call:

The matrix’s determinant is zero, so the function prints ‘Your key can not be inverted! Please restart the program.’ and bypasses the rest of the procedure.