UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

EHN 410

E-Business and Network Security

Practical 1 Guide

Original Authors: Mr. G. Naudé and Mr. A. Muller

Last updated by: Miss Elna Fourie (20 February 2025)

Due date: Thursday, 13 March 2025

Code Assignment: Submissions (AMS and Turnitin) will close at 8h30.

Practical Test: Starting @ 10h30 in the Netlabs.

1 SUBMISSION REQUIREMENTS

1.1 CODE INSTRUCTIONS

A zero mark will be awarded if the submitted code file does not run, produces errors, or does not follow the instructions.

The submission slots on the AMS and Clickup for this *Code Assignment* will be closing two hours before the start of the *Practical Test* on Thursday (13 March 2025).

No late submissions will be allowed after that point.

Everyone is required to submit code conforming to the requirements listed below:

a. All code must be commented to a point where the implementation of the underlying algorithm can be determined.

b. Your name and student number must be included as comments at the top of the submitted code file.

c. Code must be submitted via the EHN 410 ClickUp page *and* AMS. Do not email the code to the lecturer or Assistant Lecturer as the emailed code will be considered not to have been submitted.

d. The ClickUp submission must be a PDF document without a cover page including, just the code.

1. The code submission on AMS must be a single file, called StudentNumber_Prac_2.py, for example, 12345678_Prac_2.py.

e. All print statements *and helper functions* (that were not supplied within this document) used for unit testing, must be removed before submission.

f. All code must be written using Python 3.8 or newer.

g. The code submitted must be the final implementation.

h. No late assignments will be accepted. No excuses for late submission will be accepted.

i. Everyone must do their work. **Academic dishonesty is unacceptable and cases will be reported to the University Legal Office for suspension.**

1.2 SUBMISSION INSTRUCTIONS

The due date is Thursday, 13 March 2025.

Please note:

a. Code Assignment: Submission slots closing @ 8h30.
   i. Submit your final StudentNumber_Prac_1.py file on the AMS before 8h30.
   ii. Submit a *.pdf of your code on Turnitin, before 8h30.
   iii. Please upload your Turnitin Receipt onto the AMS submission slot before 8h30.

b. Practical Test: Starts @ 10h30 in the Netlabs.
   i. Written during the Practical session time slot, from 10h30 to 12h30.
   ii. Please keep the 12h30 to 13h30 slot free, in case of any technical difficulties (i.e. load-shedding, etc.).
   iii. Arrive at least 15 minutes before the test starts (10h15).
   iv. It is a coding test, based on the algorithms implemented within the *Code Assignment*.

1.3 ADDITIONAL NOTES

Only the following modules may be imported into your Python implementation file:

a. *Numpy* for data structures such as matrices and arrays.

b. *String* for string manipulation consistency.

Also, here is a simple example of what *key conversion* looks like, using the key *"CATS"*:

a. $\begin{bmatrix} C & A \\ T & S \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 0 \\ 19 & 18 \end{bmatrix}$, where *A=0, B=1, C=2, ..., Z=25*.

2 SCENARIO

Recently, there has been a surge of small startup companies looking to work from decentralized locations. Not only do they require more sophisticated communication technologies, but also better online security systems. Since some nefarious individuals or organizations constantly look to take advantage of anyone online for their benefit, the need for greater online security and privacy cannot be overlooked.

One company, in particular, is looking to develop its online communication app. This is so their employees can work from anywhere in the world, and privately communicate and collaborate with their colleagues. You have been selected as part of a small tech team that will develop a cryptographic conversion module for this communication app.

3 TASK

Working alone, you are tasked with developing a cryptographic conversion module for a private online communication app.

Your module needs to have the following functionality:

a. Each module must be able to use the Playfair cipher algorithm to encrypt and decrypt both text and individual image files (numpy arrays).

b. Each module must be able to use the Hill cipher algorithm to encrypt and decrypt both text and individual image files (numpy arrays).

c. Each module must be able to use up to a two-stage row transposition cipher algorithm to encrypt and decrypt text-based encryption keys only.

Your cryptographic conversion module will be a slot-in component to a greater system. The company management has decided to supply only the most necessary information to their engineering teams, to improve project confidentiality. However, they have disclosed the needed function calls they will use to integrate with your module.

3.1 PLAYFAIR CIPHER

Please complete the following functions:

3.1.1 playfair_get_key(isText: bool, key: str) → np.ndarray:

Used by both the text and image encryption/decryption functions; differentiate between the use cases using the Boolean variable *isText*.

For text:

a. Returns the encryption matrix used during encryption/decryption as ndarray with type String.

b. Remove duplicate values, special characters, and numbers from the key.

c. Capital letters should be changed to lowercase letters.

d. Replace 'j' with 'i'.

e. Only uses the lowercase English alphabet ('a' – 'z', excluding 'j').

For images:

a. Returns the encryption matrix used during encryption/decryption as ndarray with type int.

b. Use numbers 0 – 255.

c. Key can be any text, including uppercase, lowercase, numbers, and special characters.

d. Remove duplicate values from the key.

3.1.2 playfair_get_pos_in_key(val, val2, keyMat: np.ndarray) → np.ndarray:

a. val1 and val2 are two values that must be located in the key matrix.

b. Function returns a 1D ndarray: [val1 row, val1 col, val2 row, val2 col]

3.1.3 playfair_get_encryption_pos(pos: np.ndarray, keyMat: np.ndarray) → np.ndarray:

a. pos is the ndarray returned by pf Get Pos In Key.

b. This function applies the encryption rules of the Playfair cipher and returns a 1D ndarray with the same format as pf Get Pos In Key with the updated positions.

3.1.4 playfair_get_decryption_pos(pos: np.ndarray, keyMat: np.ndarray) → np.ndarray:

a. pos is the ndarray returned by pf Get Pos In Key.

b. This function applies the decryption rules of the Playfair cipher and returns a 1D ndarray with the same format as pf Get Pos In Key with the updated positions.

3.1.5 playfair_preprocess_text(plaintext: str) → str:

a. This function changes uppercase letters to lowercase letters and removes special characters and numbers.

b. Replace 'j' with 'i'.

c. This function adds padding and filler values to the plaintext as required. Use 'x' for this purpose.

3.1.6 playfair_encrypt_text(plaintext: str, key: str) → str:

a. This function applies the Playfair encryption cipher to the supplied string plaintext using the key.

b. Use 'x' as the padding value and filler value.

c. You can assume the supplied plaintext will not contain the letter 'x'.

3.1.7 playfair_decrypt_text(ciphertext: str, key: str) → str:

a. This function applies the Playfair decryption cipher to the supplied string plaintext using the key.

b. Remove any added padding or filler values.

3.1.8 playfair_preprocess_image(plaintext: np.ndarray) → np.ndarray:

a. This function receives the 3D image ndarray as input, flattens it into a 1D array, and adds padding and filler values as required.

b. After flattening, replace all occurrences of the value 129 with 128 and use 129 as the padding and filler value.

c. Returns the 1D padded array.

3.1.9 playfair_remove_image_padding(plaintextWithPadding: np.ndarray) → np.ndarray:

a. This function removes all occurrences of the filler value and returns the resulting 1D ndarray.

3.1.10 playfair_encrypt_image(plaintext: np.ndarray, key: str) → np.ndarray:

a. This function applies the Playfair encryption cipher to the image and returns the encrypted 1D ndarray.

b. Use 129 as the filler and padding value.

3.1.11 playfair_decrypt_image(removePadding: bool, ciphertext: np.ndarray, key: str) → np.ndarray:

a. This function applies the Playfair decryption cipher to the image and returns the encrypted 1D ndarray.

b. The function returns the decrypted and unpadded 1D image array.

c. You should not change the 128 values back into 129.

3.1.12 playfair_convert_to_image(imageData: np.ndarray, originalShape: tuple) → np.ndarray:

a. This function transforms the image data back into a 3D ndarray for displaying the image.

b. The function needs to apply some padding to convert the encrypted image array into a 3D array. How you do this is up to you.

c. After calling this function on the decrypted image array, the resulting image should be identical to the original image, except for the 129 values that were changed to 128.

3.2 HILL CIPHER

Please complete the following functions:

3.2.1 hill_get_key(isText:bool, key: str) → np.ndarray:

Used by both the text and image encryption/decryption functions; differentiate between the use cases using the Boolean variable *isText*.

For both text and images:

a. This function creates the key matrix for the hill cipher.

b. Values can repeat.

c. You may assume the key length will be 4 or 9 (returns a 2x2 or 3x3 matrix).

d. Key matrices need to be populated row by row. For example, the key ”abcd” will produce: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

e. Return an array of the correct size which consists of -1 if the key does not have a valid inverse.

For text:

a. Only the lowercase English alphabet (no special characters or numbers).

For images:

a. The key can consist of any printable ASCII value.

3.2.2 hill_get_inv_key(isText: bool, keyMat: np.ndarray) → np.ndarray:

Used by both the text and image encryption/decryption functions; differentiate between the use cases using the Boolean variable *isText*.

For both text and images:

a. This function calculates and returns the modular multiplicative inverse of the key matrix for keys used when encrypting strings.

3.2.3 hill_process_group(isText: bool, group: np.ndarray, keyMat: np.ndarray) → np.ndarray:

Used by both the text and image encryption/decryption functions; differentiate between the use cases using the Boolean variable *isText*.

For both text and images:

a. This function receives a group of numbers (with the group size being the size required by the key) and multiplies the group with the key.

b. The result is returned as a 1D ndarray of the same size as the group.

3.2.4 hill_pre_process_text(plaintext: str, keyLength: int) → np.ndarray:

a. This function receives a string message removes all special characters and numbers and changes uppercase letters to lowercase.

b. Padding is added where necessary.

c. The string is converted into a 1D ndarray of integers representing the characters (a = 0 . . . z = 25).

d. Use ‘x’ as a padding value.

3.2.5 hill_encrypt_text(plaintext: str, key: str) → str:

a. Encrypts the plaintext message using the key supplied and the Hill cipher algorithm.

b. You may assume that the key will have length = 4 or 9 for a 2x2 or 3x3 matrix respectively.

c. Returns ”Invalid Key” if the key does not have a valid inverse.

3.2.6 hill_decrypt_text(ciphertext: str, key: str) → str:

a. Decrypts the plaintext message using the key supplied and the Hill cipher algorithm.

b. Remove any added padding values.

c. Returns ”Invalid Key” if the key does not have a valid inverse.

3.2.7 hill_pre_process_image(plaintext: np.ndarray, keyLength: int) → np.ndarray:

a. This function flattens the plaintext array.

b. All occurrences of 129 are replaced with 128.

c. Padding is applied as necessary. Use 129 as the padding value.

d. Return a padded 1D ndarray.

3.2.8 hill_encrypt_image(plaintext: np.ndarray, key: str) → np.ndarray:

a. Encrypts the plaintext image using the key supplied and the Hill cipher algorithm.

b. You may assume that the key will have length = 4 or 9 for a 2x2 or 3x3 matrix respectively.

c. Use 129 as the padding value.

3.2.9 hill_decrypt_image(ciphertext: np.ndarray, key: str) → np.ndarray:

a. Decrypts the ciphertext image using the key supplied and the Hill cipher algorithm.

b. You may assume that key will have length = 4 or 9 for a 2x2 or 3x3 matrix respectively.

c. All padding should be removed.

d. You should not change the 128 values back into 129.

3.2.10 hill_convert_to_image(imageData: np.ndarray, originalShape: tuple) → np.ndarray:

a. This function transforms the image data back into a 3D ndarray for displaying the image.

b. The function needs to apply some padding to convert the encrypted image array into a 3D array. How you do this is up to you.

c. After calling this function on the decrypted image array, the resulting image should be identical to the original image, except for the 129 values that were changed to 128.

3.3 ROW TRANSPOSITION CIPHER

Please complete the following functions:

3.3.1 row_gen_key(key: str) → np.ndarray:

a. This function converts the key into a list that contains the order of the indices to use during encryption.

b. Duplicate values in the key should be removed.

c. Any printable ASCII character can be used in the key.

d. Example: Input key “azAZ!@12” should return [4 6 7 5 2 3 0 1].

e. The output array contains, in order, the indices in the key that need to be used.

   i. To clarify: The ASCII values for the key in the example (azAZ!@12) are a = 97, z = 122, A = 65, Z = 90, ! = 33, ...etc.

   ii. From there, the lowest number is 33, the '!'. The position of '!' in the original key in index 4. The second smallest number is 49, the '1'. The position of the '1' in the key is index 6. The highest number is 122, the 'z'. The position of 'z' in the key is index 1.

   iii. This is how the output of [4 6 7 5 2 3 0 1] is found. This means that the first column to use during encryption is the 4th column, the second is the 6th, and so on.

3.3.2 row_pad_text(plaintext: str, key: np.ndarray) → str:

a. This function pads that plaintext until the length is sufficient.

b. Use 'x' as the padding value.

3.3.3 row_encrypt_single_stage(plaintext: str, key: np.ndarray) → str:

a. This function applies one round of encryption to the input plaintext.

b. Returns the encrypted string.

3.3.4 row_decrypt_single_stage(ciphertext: str, key: np.ndarray) → str:

a. This function applies one round of decryption to the input ciphertext.

b. Returns the encrypted string.

3.3.5 row_encrypt(plaintext: str, key: str, stage: int) → str:

a. Uses a row transposition cipher algorithm to encrypt the plaintext message, using the provided key, and the number of stages in stage.

b. You may assume that Stage will be either 1 or 2.

c. You may assume that this function will only receive string plaintext messages.

d. If stage is greater than 1, the same key should be used to complete the higher-stage transposition process.

e. Any printable ASCII character can be used for the plaintext and the key.

f. Use 'x' as a padding value.

g. Returns the encrypted ciphertext as a string of characters.

3.3.6 row_decrypt(ciphertext: str, key: str, stage: int) → str:

a. Uses a row transposition cipher algorithm to decrypt the ciphertext message, using the provided key, and the number of stages in stage.

b. You may assume that Stage will be either 1 or 2.

c. If stage is greater than 1, the same key should be used to complete the higher-stage transposition process.

d. Both stage and key can be assumed to stay in sync with row Encrypt

e. Remove any added padding values.

f. Returns the decrypted plaintext as a string of characters.

GOOD LUCK AND HAVE FUN!

EMAIL ANY QUERIES TO: U19049910@TUKS.CO.ZA

THE ASSISTANT LECTURER IS ALSO AVAILABLE FOR CONSULTATION DURING THE TUTORIAL SESSION, EVERY FRIDAY.