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 2 Guide

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

Last updated by: Miss Elna Fourie (31 March 2025)

Due date: Tuesday, 29 April 2025

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

Practical Test: Starting @ 10h30 in the Netlabs.

(Please let me know ASAP if the test clashes with ESP 411!)

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 Tuesday (29 April 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 Tuesday, 29 April 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.

2 SCENARIO

During Practical 1, you were tasked with creating a slot-in module that contained three encryption functions. The company this was developed for, has raised a few concerns about the level of security and privacy these modules offer and have stated that they must be revisited.

The company therefore tasked you with developing an updated encryption module that uses AES, DES, and an RC4 stream cipher to perform encryption and decryption.

3 TASK

Working alone, you are tasked with developing cryptographic conversion modules for a private online communication app. Your module needs to have the following functionality:

- Each module must be able to use the 256-bit Advanced Encryption Standard (AES) algorithm to encrypt and decrypt both text and individual image files (numpy arrays).
- Each module must be able to use the Data Encryption Standard (DES) algorithm to encrypt and decrypt both text and individual image files (numpy arrays).
- Each module must be able to use the RC4 Stream Cipher algorithm to encrypt and decrypt both text and individual image files (numpy arrays).

Your cryptographic conversion module will be a slot-in component to a greater system. The company management has decided only to supply the necessary information to their engineering teams, thus improving project confidentiality further. However, they have disclosed the necessary function calls they will use to integrate with your module. Your cryptographic conversion module must have the following function definitions to be successfully integrated into the private communication app (They may not be altered):

3.1 AES CIPHER

3.1.1 aes_Generate_Round_Keys(key: str, sBox: hex ndarray) → int ndarray:

- Returns an ndarray containing all of the round keys required for the 256-bit AES cipher.
- The key can consist of any printable ASCII character.
- You may assume the key length will be 32 bytes.
- Each round key is a 4x4 int ndarray (ex., [[[97 101 105 109]...).
- sBox is a 16x16 ndarray, which consists of hex values (ex., ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76'])

3.1.2 aes_Preprocess_String_Plaintext(plaintext: str) → 1D int ndarray:

- This function pads the plaintext to a length divisible by 16.
- The plaintext can be any printable ASCII characters.
- The padding scheme to use is called PKCS5. It works by padding the plaintext with the number of padding bytes required, with the padding byte equal to the number of padding bytes required. For example, if 4 padding bytes are required, the padding will be (in hex) '04040404'. If only two padding bytes are required, the padding will be '0202'. If 12 Padding bytes are required, the padding will be '0C0C0C0C0C0C0C0C0C0C0C0C'.
- Padding is always added, even if the padding added is a full block.
- The function returns a 1D int ndarray, which consists of the int representation of the plaintext with the padding added, (ex., [72 105 14 14 14...).

3.1.3 aes_Create_Input_States(inputBytes: 1D int ndarray) → int ndarray:

- Returns an int ndarray containing all of the initial states that will be encrypted.
- inputBytes is a 1D int array, that is returned by aes_Preprocess_String_Plaintext or aes_Preprocess_Image_Plaintext.
- An initial state is a 16-byte block of data that will be encrypted, and formatted as a 4x4 int ndarray.
- Return shape will be (x, 4, 4), where x is the number of 16-byte blocks in the padded plaintext, (ex., [[[97, 97, 97, 4], [97, 97, 97, 4], [97, 97, 97, 4], [97, 97, 97, 4]]], shape: (1,4,4)).

3.1.4 aes_remove_Padding(paddedArray: 1D int ndarray) → 1D int ndarray:

- The function removes the padding from the input array, and returns the resulting int ndarray (ex., [72 105]).
- This function can be used for text and images.

3.1.5 aes_Encrypt_String(plaintext: str, key: str) → int ndarray:

- This function encrypts the input plaintext and returns the ciphertext as an int ndarray (ex., [17 220 23 45...].
- Both the plaintext and key can be any printable ASCII characters.
- The function loads the sBox from an npy file that will be provided. Make sure you do not change the included line of code that loads the file.

3.1.6 aes_Decrypt_String(ciphertext: int ndarray, key: str) → str:

- This function decrypts the input ciphertext array and returns the plaintext as a string (ex. ”Hi”).
- Padding should be removed.
- The function loads the sBox and inverse sBox from the npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.1.7 aes_Preprocess_Image_Plaintext(plaintext: 3D int ndarray) → 1D int ndarray:

- This function flattens the input plaintext array and returns a 1D int ndarray consisting of the flattened plaintext and added padding.
- The same PKCS5 padding scheme must be used.
- Padding is always added, even if the padding added is a full block.

3.1.8 aes_Encrypt_Image(plaintext: 3D int ndarray, key: str) → 1D int ndarray:

- This function encrypts the input image plaintext and returns the ciphertext as an int ndarray (ex., [17 220 23 45...).
- The key can be any printable ASCII characters.
- The function loads the sBox from an npy file that will be provided. Make sure you do not change the included line of code that loads the file.

3.1.9 aes_Decrypt_Image(ciphertext: int ndarray, key: str) → 1D int ndarray:

- This function decrypts the input ciphertext image array and returns the plaintext as a 1D int array (ex., [17 220 23 45...).
- Padding should be removed.
- The function loads the sBox and inverse sBox fromthe npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.1.10 aes_Add_Round_key(state: 2D int ndarray, roundKey: 2D int ndarray) → 2D int ndarray:

- This function applies the add round key transformation to the state.
- The state, roundKey, and return value are all 2D 4x4 int ndarrays (ex. [[1 2 3 4], [...]).
- This function will be used during encryption and decryption.

3.1.11 aes_Substitute_Bytes(state: 2D int ndarray, sBox: 2D hex ndarray) → 2D int ndarray:

- This function applies the substitute bytes transformation to the state.
- The state and return value are both 2D 4x4 int ndarrays (ex. [[1 2 3 4], [...]).
- sBox is a 16x16 ndarray, which consists of hex values (ex., ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76']).
- This function will be used during encryption and decryption.

3.1.12 aes_Shift_Rows_Encrypt(state: 2D int ndarray) → 2D int ndarray:

- This function applies the shift rows transformation to the state.
- The state and return value are both 2D 4x4 int ndarrays (ex., [[1 2 3 4], [...]).

3.1.13 aes_Shift_Rows_Decrypt(state: 2D int ndarray) → 2D int ndarray:

- This function applies the inverse shift rows transformation to the state.
- The state and return value are both 2D 4x4 int ndarrays (ex., [[1 2 3 4], [...]).

3.1.14 aes_Mix_Columns_Encrypt(state: 2D int ndarray) → 2D int ndarray:

- This function applies the mix columns transformation to the state.
- The state and return value are both 2D 4x4 int ndarrays (ex., [[1 2 3 4], [...]).

3.1.15 aes_Mix_Columns_Decrypt(state: 2D int ndarray) → 2D int ndarray:

- This function applies the inverse mix columns transformation to the state.
- The state and return value are both 2D 4x4 int ndarrays (ex., [[1 2 3 4], [...]).

3.1.16 aes_Apply_Encryption_Round(state: 2D int ndarray, roundKey: 2D int ndarray, sBox: 2D hex ndarray) → 2D int ndarray:

- This function applies a single encryption round to the input state.
- A single round consists of all of the following transformations in the order: sub bytes, shift rows, mix cols, and add round key.
- The state, roundKey and return value are all 2D 4x4 int ndarrays (ex., [[1 2 3 4], [...]).
- sBox is a 16x16 ndarray, which consists of hex values (ex., ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76'])

3.1.17 aes_Encrypt_State(state: 2D int ndarray, roundKeys: 3D int ndarray, sBox: 2D hex ndarray) → 2D int ndarray:

- This function applies the full encryption to the input state.
- The state and return value are both 2D 4x4 int ndarrays (ex. [[1 2 3 4], [...]).
- sBox is a 16x16 ndarray, which consists of hex values (ex., ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76'])
- roundKeys is the return value of the aes_Generate_Round_Keys function.

3.1.18 aes_Apply_Decryption_Round(state: 2D int ndarray, roundKey: 2D int ndarray, sBox: 2D hex ndarray) → 2D int ndarray:

- This function applies a single decryption round to the input state.
- A single round consists of all of the following transformations in the order: inv shift rows, inv sub bytes, add round key, and inv mix cols.
- The state, roundKey and return value are all 2D 4x4 int ndarrays (ex. [[1 2 3 4], [...]).
- sBox is a 16x16 ndarray, which consists of hex values (ex. ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76'])

3.1.19 aes_Decrypt_State(state: 2D int ndarray, roundKeys: 3D int ndarray, sBox: 2D hex ndarray) → 2D int ndarray:

- This function applies the full decryption to the input state.
- The state and return value are both 2D 4x4 int ndarrays (ex. [[1 2 3 4], [...]).
- sBox is a 16x16 ndarray, which consists of hex values (ex. ['0x63' '0x7C' '0x77' '0x7B' '0xF2' '0x6B' '0x6F' '0xC5' '0x30' '0x01' '0x67' '0x2B' '0xFE' '0xD7' '0xAB' '0x76'])
- roundKeys is the return value of the aes_Generate_Round_Keys function.

3.1.20 aes_des_rc4_Convert_To_Image(arrayToConvert: 1D int ndarray, originalShape: tuple) → 3D int ndarray:

- This function transforms the image data back into a 3D ndarray for displaying the image.
- originalShape is the tuple returned from np.shape of the original image before encryption.
- The function will need to apply some kind of padding to be able to convert the encrypted image array into a 3D array. How you do this is up to you. Do not simply reshape the array back to the original shape for an encrypted image. You are throwing away data and that is never good.
- After calling this function on the decrypted image array, the resulting image should be identical to the original image.

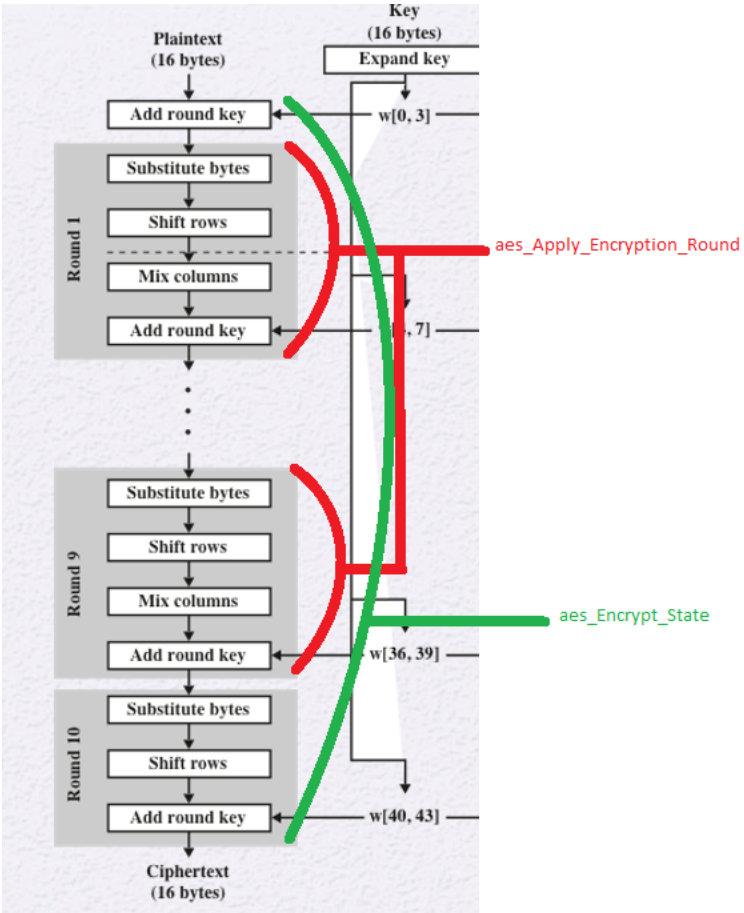

Figure 1: Difference between aes_Apply_Encryption_Round and aes_Encrypt_State.

3.2 DES CIPHER

3.2.1 des_Generate_Round_Keys(key: str, permutedChoice1: ndarray, permutedChoice2: ndarray, roundShifts: ndarray) → ndarray:

- The function generates the round keys for the DES algorithm.
- The function returns a 1D hex string array containing the 48-bit round keys (ex. ['E0BE66132A82' 'E0B676102307' ...).
- The key can consist of any printable ASCII characters.
- You may assume the key will be 8 bytes.
- permutedChoice1 is a 1D int ndarray (ex, [57 49 41 33 ...).
- permutedChoice2 is a 1D int ndarray (e,x [14 17 11 24 ...).
- roundShifts is a 1D int ndarray that contains the number of shifts for each round.

3.2.2 des_Preprocess_String_Plaintext(plaintext: str) → 1D hex ndarray:

- The function returns a 1D hex ndarray that contains the hex values of the plaintext and added padding (ex,. ['48' '69' '06' '06' '06' '06' '06' '06']).
- The plaintext can be any printable ASCII characters.
- Padding is always added, even if the padding added is a full block.
- The padding should be the PKCS5 scheme, padded to block lengths of 8 bytes.

3.2.3 des_Create_Input_Blocks(processedArray: 1D hex ndarray) → 1D hex string ndarray:

- The function returns a 1D hex string ndarray that consists of 8 bytes in hex format. These are the blocks to encrypt (ex., ['4869060606060606'])
- processedArray is the array returned by des_Preprocess_String_Plaintext.

3.2.4 des_Remove_String_Padding(paddedArray: 1D hex ndarray) → 1D hex ndarray:

- The function returns a 1D hex ndarray that contains the plaintext without padding (ex. ['48' '69']).

3.2.5 des_Encrypt_String(plaintext: str, key: str) → 1D hex ndarray:

- The function encrypts a string plaintext and returns it as a 1D hex array (ex. ['C0' '6A' '85' '57' 'A0' '95' '4D' '5B']).
- plaintext and key can be any printable ASCII characters.
- The function loads various permutation tables from npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.2.6 des_Decrypt_String(ciphertext: 1D hex ndarray, key: str) → str:

- The function decrypts the given ciphertext ndarray and returns it as a str (ex. ”Hi”).
- ciphertext is the 1D hex ndarray returned by des_Encrypt_String.
- The key can be any printable ASCII characters.
- Padding should be removed.
- The function loads various permutation tables from npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.2.7 des_Process_Block(block: str, roundKeys: 1D hex str ndarray, initialPerm: 1D int ndarray, sBoxes: 3D int ndarray, expansionBox: 1D int ndarray, FpermChoice: 1D int ndarray, invInitialPerm: 1D int ndarray) → str:

- This function applies the DES encryption to a single block of data (one item in the ndarray returned by des_Create_Input_Blocks).
- block is a hex str (ex ’4869060606060606’).
- The function returns the hex string block (ex ’4869060606060606’).
- roundKeys are the round keys generated by des_Generate_Round_Keys for encryption, or the reverse thereof for decryption.
- initialPerm is the initial permutation and is a 1D int ndarray (ex [58 50 42 34 26...).
- sBoxes is a 3D int ndarray with shape (8, 4, 16). They are used in the F-Function of the DES algorithm.
- expansionBox is a 1D int ndarray used in the F-function.
- FpermChoice is the permutation choice used in the F-Function. This is a 1D int ndarray.
- invInitialPerm is a 1D int ndarray. This is the inverse initial permutation.

3.2.8 des_Process_Round(roundInputValue: hex str, roundKey: hex str, sBoxes: 3D int ndarray, expansionBox: 1D int ndarray, permutationChoice: 1D int ndarray) → hex str:

- This function applies a single DES encryption round of the F-Function to the hex str input.
- roundInputValue is the block to be encrypted (ex,’4869060606060606’).
- roundKey is the current round key, as a hex string (ex.,’4869060606060606’).
- The function returns the modified hex string (ex.,’4869060606060606’).
- sBoxes is a 3D int ndarray with shape (8, 4, 16).
- expansionBox is a 1D int ndarray.
- permutationChoice is a 1D int ndarray.

3.2.9 des_Preprocess_Image_Plaintext(plaintext: 3D int ndarray) → 1D hex ndarray:

- This function flattens the input plaintext array and returns a 1D hex ndarray consisting of the flattened plaintext and added padding, (ex. ['02' '00' '03' ... '04' '04' '04']).
- The same PKCS5 padding scheme must be used.
- Padding is always added, even if the padding added is a full block.

3.2.10 des_Remove_Image_Padding(paddedArray: 1D hex ndarray) → 1D int ndarray:

- The function returns a 1D int ndarray that contains the plaintext without padding (ex. [2 0 3 ... 16 21 139]).

3.2.11 des_Encrypt_Image(plaintext: 3D int ndarray, key: str) → 1D int ndarray:

- This function encrypts the input image plaintext and returns the ciphertext as an int ndarray (ex., [17 220 23 45...).
- The key can be any printable ASCII characters.
- The function loads various permutation tables from npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.2.12 des_Decrypt_Image(ciphertext: int ndarray, key: str) → 1D int ndarray:

- This function decrypts the input ciphertext image array and returns the plaintext as a 1D int array (ex., [17 220 23 45...).
- Padding should be removed.
- The function loads various permutation tables from npy files that will be provided. Make sure you do not change the included lines of code that load the files.

3.2.13 des_Apply_Permutation(valueToPermute: str, permuteTable: 1D int ndarray, numBitsBeforePermute: int) → str:

- This function applies the permutation to the input.
- The function returns the permuted hex string (ex,"33BF6ED1").
- valueToPermute is the input to permute as a hex string (ex,"33BF6ED1").
- permuteTable is a 1D int ndarray that defines the permutations.
- numBitsBeforePermute is the bit length of the input before the permutation, for example during key generation, a 64-bit key is permuted to a 56-bit key; thus, in this case, numBitsBeforePermute will be 64.

3.2.14 des_Split_In_Two(inputValue: str) → 1D hex str ndarray:

- This function splits the input into two halves.
- inputValue is the hex string value that needs to be split (ex,"E4600FA647F7C412").
- The function returns a 1D hex str ndarray containing the two halves of the input (ex., ['E4600FA6' '47F7C412']).

3.2.15 des_XOR(value1: hex str, value2: hex str) → hex str:

- This function returns the XOR result of the two input hex values as a hex string (ex., "C01CDB70CB0F").
- value1 and value2 are hex strings (ex., "C01CDB70CB0F")

3.2.16 des_left_Shift(inputValue: str, shiftCount: int) → str:

- This function performs a circular bit shift on the input array.
- The array is circularly shifted shiftCount times.
- The input and return types are hex string, e.g.: '47F7C412'

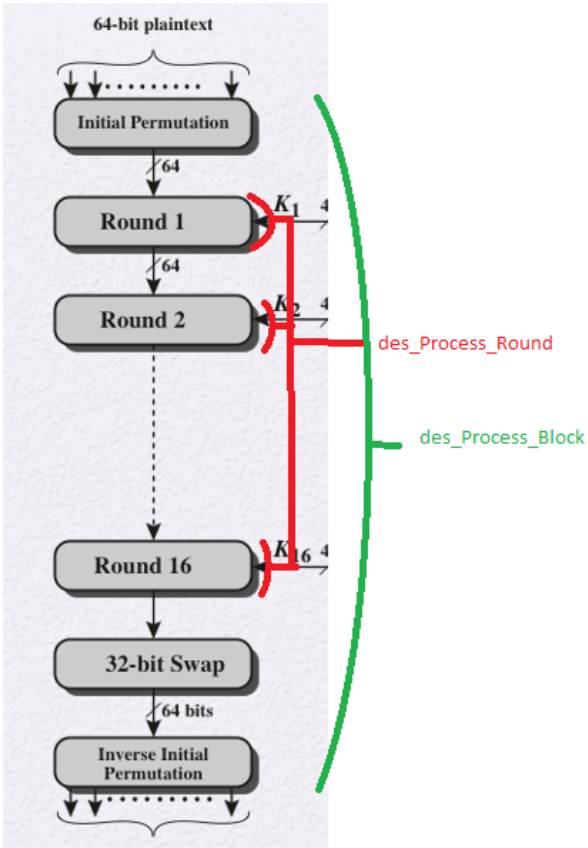

Figure 2: Difference between des_Process_Round and des_Process_Block.

3.3 RC4 STREAM CIPHER

3.3.1 rc4_Init_S_T(key: str) → 2D int ndarray:

- This function creates the initial S and T arrays for the RC4 stream cipher.
- key can be any printable ASCII characters.
- The function returns an int ndarray, that contains the S int ndarray in the first index and the T int ndarray in the second index.
- Example: S = [1 2 3], T = [5 3 6]. Function return [S, T] → [[1 2 3] [5 3 6]]

3.3.2 rc4_Init_Permute_S(sArray: 1D int ndarray, tArray: 1D int ndarray) → 1D int ndarray:

- This function performs the initial permutation on the S array.
- The function returns the permuted S 1D int ndarray (ex., [102 115 41 ...).

3.3.3 rc4_Generate_Stream_Iteration(i: int, j: int, sArray: 1D int ndarray) → tuple:

- This function generates a random stream of bytes.
- i and j are values used during the stream generation.
- sArray is the last modified S array.
- The function returns a tuple with the following structure: (modified i, modified j, modified S array, generated k stream byte).
- Example: (2, 156, [102 196 83 ... 63], 7)

3.3.4 rc4_Process_Byte(byteToProcess: int, k: int) → int:

- This function returns the XOR result between the byteToProcess and k inputs.

3.3.5 rc4_Encrypt_String(plaintext: str, key: str) → 1D int ndarray:

- This function encrypts the input plaintext and returns a 1D int ndarray containing the result.
- No padding is required.
- The plaintext and key can be any printable ASCII characters.

3.3.6 rc4_Decrypt_String(ciphertext: 1D int ndarray, key: str) → str:

- This function decrypts the input ciphertext array and returns a string containing the plaintext.
- The key can be any printable ASCII characters.

3.3.7 rc4_Encrypt_Image(plaintext: 3D int ndarray, key: str) → 1D int ndarray:

- This function flattens the image input ndarrays, encrypts the flattened ndarray, and returns a 1D int ndarray containing the result.
- No padding is required.
- The key can be any printable ASCII characters.

3.3.8 rc4_Decrypt_Image(ciphertext: 1D int ndarray, key: str) → 1D int ndarray:

- This function decrypts the input ciphertext image ndarray and returns a 1D int ndarray containing the plaintext image data.
- The key can be any printable ASCII characters.