National College of Ireland

Configuration Manual

MSc Research Project
Cyber Security

Kevin Kehoe

Student ID: x20147228

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kevin Kehoe
Student ID:	x20147228
Programme:	Cyber Security
Year:	2022
Module:	MSc Research Project
Supervisor:	Ross Spelman
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1956
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	*Kevin Kehoe*
Date:	12th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	✓□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kevin Kehoe
x20147228

1 Introduction

In my paper Secure Cryptography Algorithm using Rubik's Cube for IOT Devices, I proposed an encryption algorithm based around a 4x4 Rubik's cube and the AES algorithm. This configuration manual will describe the steps that need to be taken to setup the environment on a Raspberry Pi 4 Model B and allow the tests to be conducted. Some of the code will be explained in this configuration manual to demonstrate how this algorithm runs.

2 Environment Setup

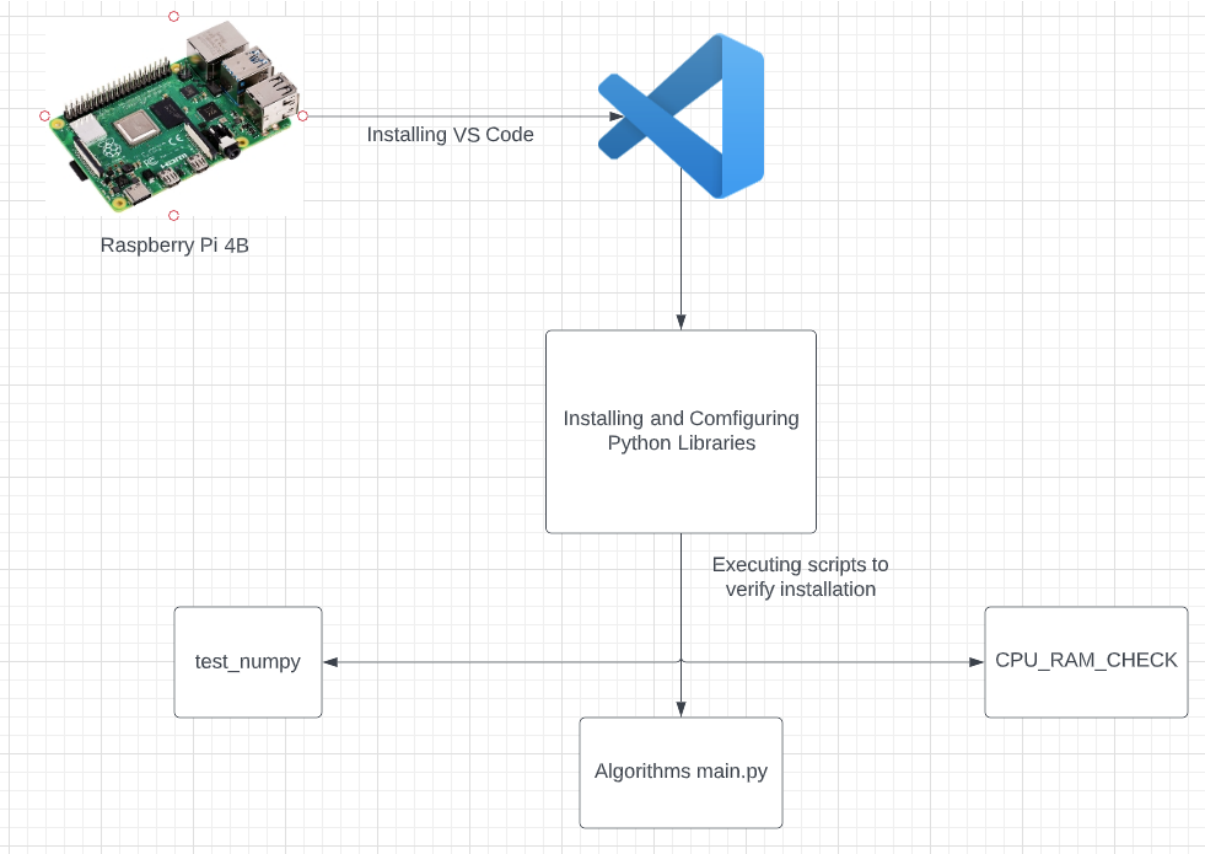
The research question is to propose a new cryptography algorithm that can work on IoT devices without affecting performance. This will be achieved by comparing the results from the proposed algorithm against a Python lightweight algorithm chosen by the NIST and comparing Python implementations of more know algorithms such as AES. All these algorithms will be evaluated on the same environment so that the results of these tests will be accurate.

Section 2 will give detailed instructions for installing each of the following requirements.

- Installing the Raspberry PI OS on the Raspberry Pi Model 4B
- Updating the packages on the Raspberry Pi OS
- Installing and configuring settings inside of VSCode for optimal performance
- Installing Python and other Python Libraries. Scripts will be included to test that the installations were successful

In addition, particular aspects of the proposed algorithm will be highlighted to indicate its uniqueness in comparison to current algorithms.

2.1 Diagram of Environment Setup



2.2 Installing the Raspberry Pi OS

To install the Raspberry Pi OS onto the Raspberry Pi Model 4B, the Raspberry Pi imager needs to be downloaded from Raspberry Pi's official site. In order to use this software, a Micro SDHC card with the capacity of 32GB's is required, as well as a Micro SDHC card reader. Click on the "Download for Windows" option and install the exe file downloaded. With the Micro SDHC card inserted into the Micro SDHC card reader, plug the reader into the computer and run the Raspberry Pi Imager. Next is to choose the Operating System that will be installed onto the Raspberry Pi 4B. There are a few different versions of the OS that can be installed but for this environment setup, choose the default 32-bit version.

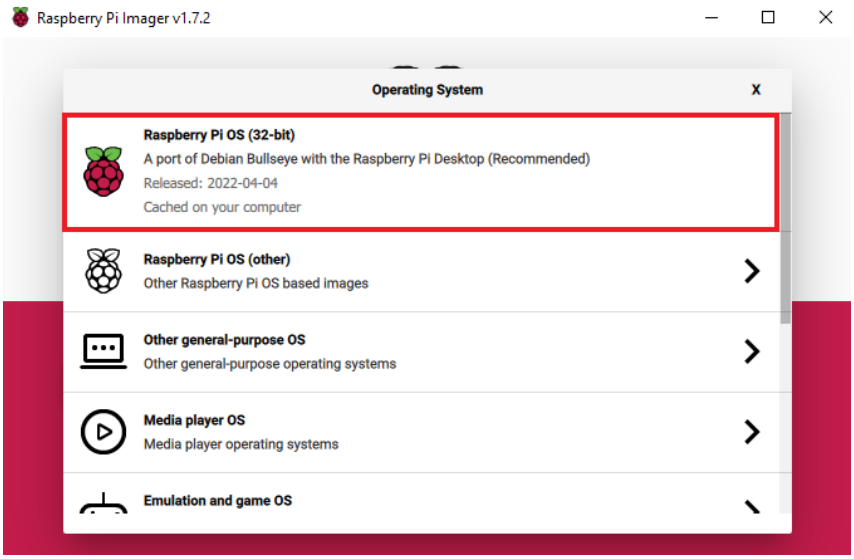


Figure 2: Network Lab Diagram

Once the storage device has been selected, click on the write option, and wait for the process to be completed. When this has been completed, the SDHC card is ready and can be inserted into the Raspberry Pi.

2.3 Configuring the Raspberry Pi Environment

After going through the initial setup, we need to execute a few commands to update all dependencies that are on the environment. Open a terminal and type the following command to update the package list and then update all the dependencies.

sudo apt-get update && sudo apt-get upgrade

With the Raspberry Pi updated, it's time to install Visual Studio Code as our Integrated Desktop Environment. Inside the terminal, execute the following command to install Visual Studio Code. This might take some time depending on your internet speed.

sudo apt install code

Once Visual Studio Code has been installed, launch it, and use the keyboard shortcut CTRL+P to open the Quick Open option. If we type in >runtime into this box, we will open the argv.json file. This file is our Runtime Arguments which can allow us to change one option to use software rendering instead of hardware accelerated rendering.

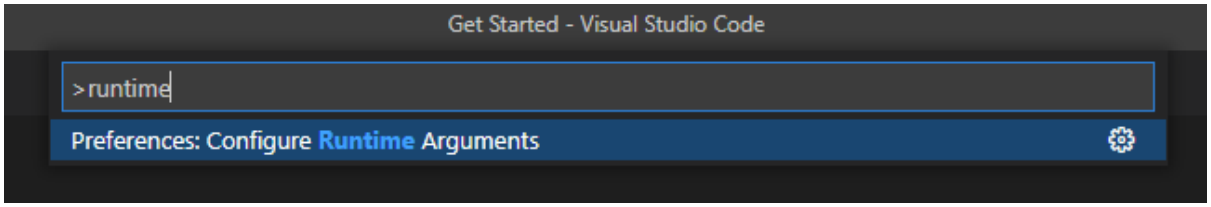


Figure 3: Locating Runtime Arguments in VSCode

The argv.json file will look like Figure 4 when loading it for the first time.

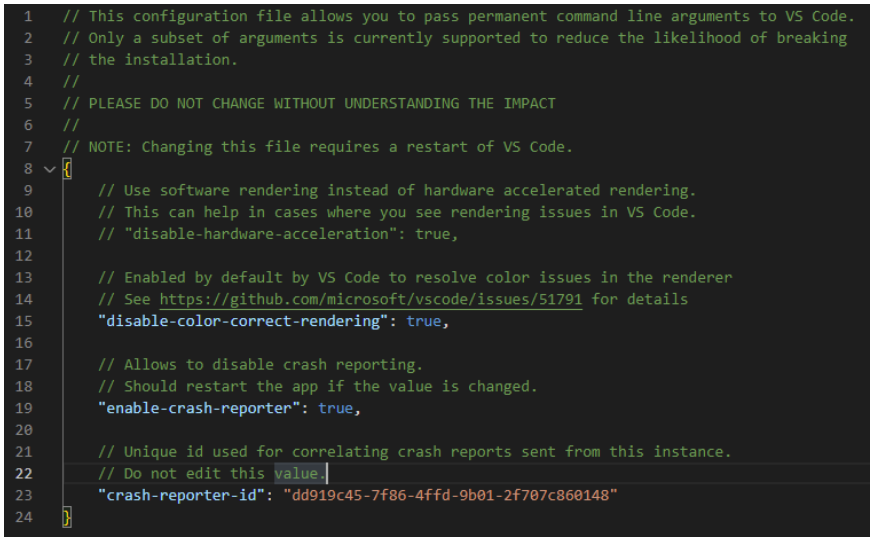


Figure 4: Argv.json before changes

If we remove the two forward slashes from line eleven, we can allow Visual Studio Code to use the software rendering method. Figure 5 shows the change that was made. After saving the change, restart Visual Studio Code.

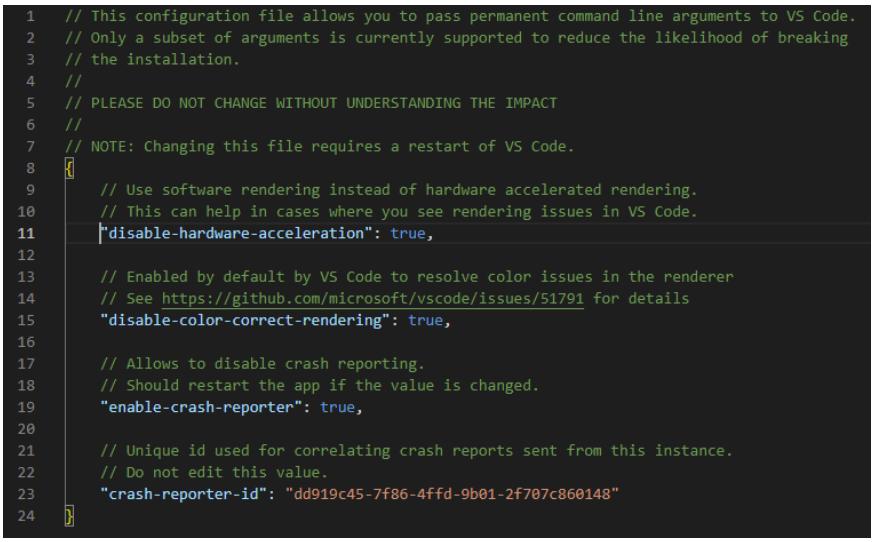


Figure 5: Argv.json after changes

2.4 Installing Python Libraries

On the Raspberry Pi OS, Python comes preinstalled with version 3.9.5. We can check this by running the command.

python -V

For the proposed algorithm, the Numpy package will be needed to run the scripts. “NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed” [1]. To install Numpy, it is recommended that

PIP3 is installed since Python 3 is installed on the OS. Running the command below will install PIP3 onto the system.

sudo apt install python3-pip

After installing PIP3, we can install Numpy which can be done with the below command.

pip3 install numpy

To assess whether the package installed successfully, run the test_numpy script in Visual Studio code. This code should split an array of six values into a 3x3 array.

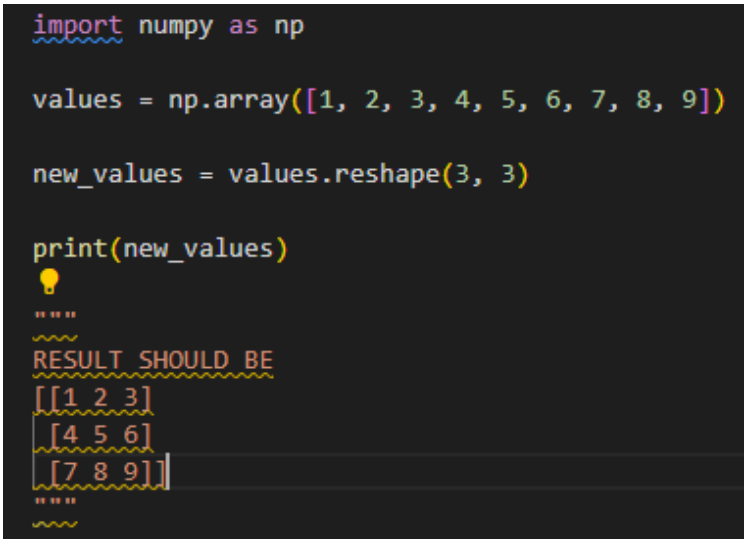


Figure 6: test_numpy script

The other requirements needed to be checked on the device when executing the algorithms scripts are the CPU and RAM usage. To check these usages on the Raspberry Pi, the packet psutil can be used. “psutil (Python system and process utilities) is a Python package that retrieves information on ongoing processes and system usage (CPU, memory, storage, network, and sensors). It is mostly used for system monitoring, profiling, restricting process resources, and process management” [4].

To install the psutil package, two commands need to be ran. The first command to run is

sudo apt-get install gcc python3-dev

This command needs to be ran because the package python-dev has some headers files that are needed for Pythons C API. Psutil requires these headers to function in our algorithm. To install the psutuil package after installing the Python C Headers, run the following command.

pip install --no-binary :all: psutil

The --no-binary :all: section of the command is to exclude any binary packages when installing a package from source. To test if psutil has been successful been installed, a test script called CPU_RAM_Check has been included in the algorithms folder so it can be run and verify it is working.

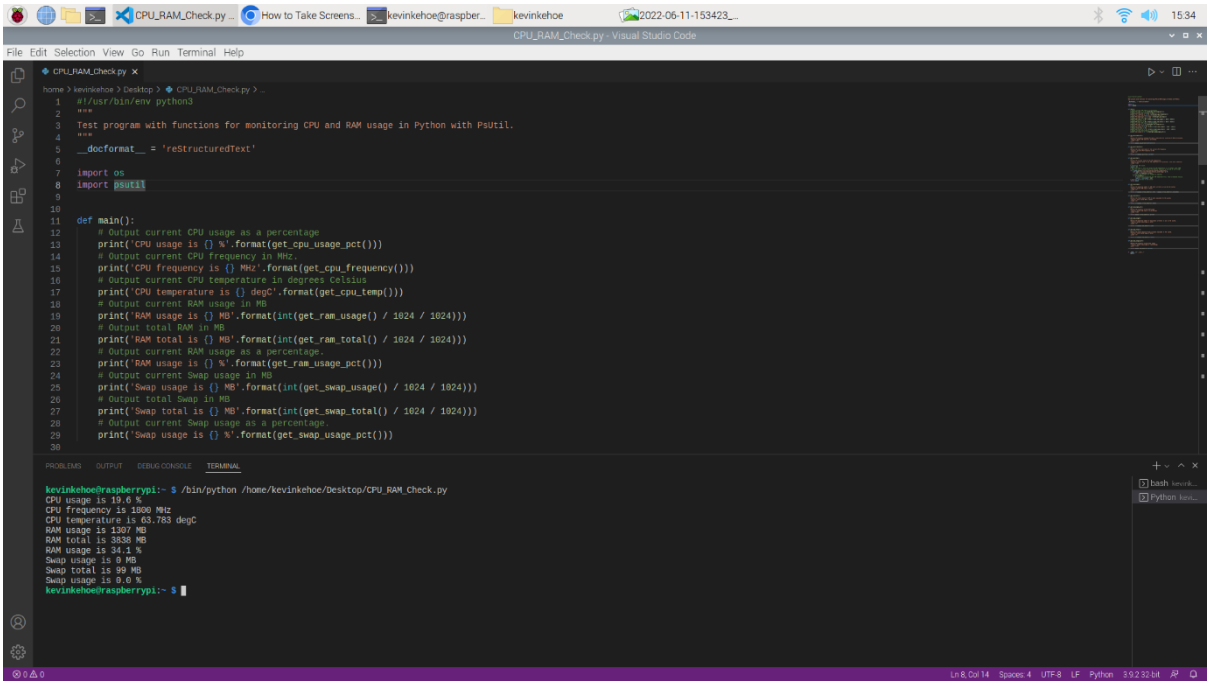


Figure 7: CPU_RAM_Check – verifying script works with results

3 Proposed Algorithm Configuration

The proposed algorithm has six files in total which are responsible for different purposes. The main three which are covered in this configuration manual are

- Main
- Functions
- Movements

3.1 Main file

The main file is responsible for running both the encrypt and decrypt methods. For the encrypt method, the plaintext and key variables must be either 16, 24 or 32 characters long each. The algorithm will not work if the length of both variables is different.

```
7   plaintext="Thistextwillwork" # <-- 16 characters long
8   key="cP9^jPQ5*du>#{<E" # <-- 16 characters long
```

Figure 7: main script – correct lengths

```
7   plaintext="Thistextwillwork" # <-- 16 characters long
8   key="cP9^jPQ5*" # <-- 8 characters long
```

Figure 8: main script – incorrect lengths

The results of both encryption and decryption are stored in the following variables to prove the algorithm works as designed.

- ciphertext
- rot_cipher_key
- decrypted_plaintext
- decrypted_rot_key

A timer has been added to the script to test the execution time of running both the encrypt and decrypt methods. The start_time method will grab the time at the beginning of the scripts execution and Figure 7 will remove that time from the current time to get the result.

```
86    print("Process finished --- %s seconds ---" % (time.time() - start_time))
```

Figure 9: main script – execution time

3.2 Functions file

The functions file is responsible for almost all the operations done by the algorithm. The key_schedule function runs on both the encrypt and decrypt methods. A key schedule takes the initial key provided and does different operations on it to create many more round keys. Depending on the size of the plaintext and key variables, a different number of rounds are performed to make more keys. A different key from this schedule is used for each round of the process. This function starts on line 10 and ends on line 149.

Since the algorithm is based on a 4x4 Rubik's cube, a create_cube was designed to make 6 4x4 arrays which would act as our cube of 96 faces. Reshaping the arrays can be achieved with NumPy which was installed earlier.

```
150   # CREATE THE 4x4x4 CUBE (Total 96 faces)
151   def create_cube():
152       up_face = np.full(16,"  ").reshape(4,4)
153       down_face = np.full(16,"  ").reshape(4,4)
154       left_face = np.full(16,"  ").reshape(4,4)
155       front_face = np.full(16,"  ").reshape(4,4)
156       right_face = np.full(16,"  ").reshape(4,4)
157       back_face = np.full(16,"  ").reshape(4,4)
158       return up_face, down_face, left_face, front_face, right_face, back_face
159
```

Figure 10: functions script – creating the cube

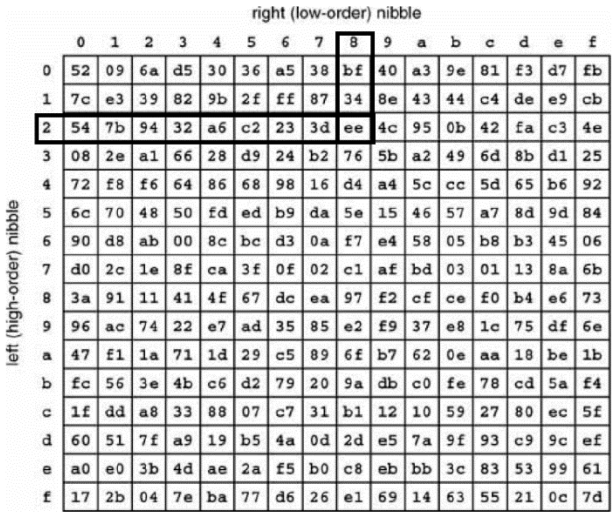
From our key_schedule results, we pass those results and the plaintext into the face_operation function. The plaintext's orientation will be altered to look like Figure 11 before performing any of the operations. After this change, certain operations will be carried out with the plaintext and a key based on which round is being performed. These operations will be one of the following depending on the round.

- xor
- sbox
- shift_rows
- mix_columns

"XOR is a bitwise operator, and it stands for "exclusive or." It performs logical operation. If input bits are the same, then the output will be false(0) else true(1)" [2]. For this operation to be carried out, both values must be converted to be hexadecimal values.

The sbox function will take our hexadecimal value and check it against a table. The first character will be checked on the left side and the second value will be checked on the right side. The sbox used in this algorithm is based on the Rijndael AES S-box. If our hexadecimal

value is 28, we use the 2 on the left side and the 8 on the right side. Where these two values intercepts are the value that we substitute 28 for, which in this case is ee.



left (high-order) nibble \ right (low-order) nibble	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 11: AES SBox – How to use

The shift_rows function will shift values in our 4x4 based on the length of the key. Using NumPys array roll functionality will move the values in the array based on the iteration of the loop. Each row is moved a different amount which is quite like how the AES algorithm applies it but shifted by different amounts.

```
def shift_rows(round_result):
    if len(round_result) == 16:
        round_result_rows = np.array_split(round_result, 4)
        for index, value in enumerate(round_result_rows):
            round_result_rows[index] = np.roll(round_result_rows[index], -index)
        round_result = []
        i, j = 0, 0
        while i < 4:
            while j < 4:
                round_result.append(round_result_rows[i][j])
                j += 1
            j = 0
            i += 1
```

Figure 12: shift_rows – snippet of shifting

The mix_columns function will add our hex values with the predefined matrix by using the l_lookup provided in the mix_columns_table file and then substitute the final value in the e_lookup table. These tables are also known as Galois Field tables [3].

```
e_table = [
    ['--', 'x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'xa', 'xb', 'xc', 'xd', 'xe', 'xf'],
    ['0x', '01', '03', '05', '0f', '11', '33', '55', 'ff', '1a', '2e', '72', '96', 'a1', 'f8', '13', '35'],
    ['1x', '5f', 'e1', '38', '48', 'd8', '73', '95', 'a4', 'f7', '02', '06', '0a', '1e', '22', '66', 'aa'],
    ['2x', 'e5', '34', '5c', 'e4', '37', '59', 'eb', '26', '6a', 'be', 'd9', '70', '90', 'ab', 'e6', '31'],
    ['3x', '53', 'f5', '04', '0c', '14', '3c', '44', 'cc', '4f', 'd1', '68', 'b8', 'd3', '6e', 'b2', 'cd'],
    ['4x', '4c', 'd4', '67', 'a9', 'e0', '3b', '4d', 'd7', '62', 'a6', 'f1', '08', '18', '28', '78', '88'],
    ['5x', '83', '9e', 'b9', 'd0', '6b', 'bd', 'dc', '7f', '81', '98', 'b3', 'ce', '49', 'db', '76', '9a'],
    ['6x', 'b5', 'c4', '57', 'f9', '10', '30', '50', 'f0', '0b', '1d', '27', '69', 'bb', 'd6', '61', 'a3'],
    ['7x', 'fe', '19', '2b', '7d', '87', '92', 'ad', 'ec', '2f', '71', '93', 'ae', 'e9', '20', '60', 'a0'],
    ['8x', 'fb', '16', '3a', '4e', 'd2', '6d', 'b7', 'c2', '5d', 'e7', '32', '56', 'fa', '15', '3f', '41'],
    ['9x', 'c3', '5e', 'e2', '3d', '47', 'c9', '40', 'c0', '5b', 'ed', '2c', '74', '9c', 'bf', 'da', '75'],
    ['ax', '9f', 'ba', 'd5', '64', 'ac', 'ef', '2a', '7e', '82', '9d', 'bc', 'df', '7a', '8e', '89', '80'],
    ['bx', '9b', 'b6', 'c1', '58', 'e8', '23', '65', 'af', 'ea', '25', '6f', 'b1', 'c8', '43', 'c5', '54'],
    ['cx', 'fc', '1f', '21', '63', 'a5', 'f4', '07', '09', '1b', '2d', '77', '99', 'b0', 'cb', '46', 'ca'],
    ['dx', '45', 'cf', '4a', 'de', '79', '8b', '86', '91', 'a8', 'e3', '3e', '42', 'c6', '51', 'f3', '0e'],
    ['ex', '12', '36', '5a', 'ee', '29', '7b', '8d', '8c', '8f', '8a', '85', '94', 'a7', 'f2', '0d', '17'],
    ['fx', '39', '4b', 'dd', '7c', '84', '97', 'a2', 'fd', '1c', '24', '6c', 'b4', 'c7', '52', 'f6', '01']
]
```

Figure 13: mix_columns – e_table

With the new ciphertext being created after the face_operation table, the random_orientation will take the ciphertext and assign values to random faces on the cube. The length of the ciphertext will always be 96 bytes long. Based on which face a hexadecimal value is stored to, a mathematical operation is performed to generate that number and store it in the orientation array. Once all the values are assigned onto the cube, the next function scramble_rot_key will be executed.

```
def random_orientation(up_face, down_face, left_face, front_face, right_face, back_face, xor, orientation):
    faces = [up_face, down_face, left_face, front_face, right_face, back_face]
    xor = np.array(xor)
    xor = xor.ravel()
    i = 0
    while i < 96:
        index, column, row = choose_random_face(faces)
        if faces[index][column][row] == "  ":
            faces[index][column][row] = xor[i]
            i += 1
            final_value = len(faces[index][column]) * 4 * index + len(faces[index][column]) * column + row
            orientation.append(final_value)
    up_face, down_face, left_face, front_face, right_face, back_face = faces[0], faces[1], faces[2], faces[3], faces[4], faces[5]
    return up_face, down_face, left_face, front_face, right_face, back_face, xor, orientation
```

Figure 14: random_orientation

The scramble_rot_key function will execute 96 different cube related movements and store them in the rotation_key_array. For each movement a random number from the orientation array will be added to the key. A possible result would be FWA237. The FWA2 is the cubes movement and the 37 indicates the position where the first value of the ciphertext is placed on the cube. It’s important to mark this position since it will be needed to place the text in the same place when decrypting.

```
def scramble_rot_key(up_face, down_face, left_face, front_face, right_face, back_face, orientation, final_key, ciphertext):
    faces = [up_face, down_face, left_face, front_face, right_face, back_face]
    rotation_key_array = []
    for i in range(96):
        index = random.randrange(0, len(movements.movements))
        up_face, down_face, left_face, front_face, right_face, back_face = movements.movements[index][1](up_face, down_face, left_face, front_face, right_face, back_face)
        rotation_key_array.append(movements.movements[index][0])

    faces = [up_face, down_face, left_face, front_face, right_face, back_face]
    random.shuffle(orientation)
    ciphertext = ciphertext_altering(orientation, ciphertext, faces)
    rotation_key_array.reverse()
    final_key = ""
    for i, v in enumerate(rotation_key_array):
        final_key = final_key + v + str(orientation[i])
    return up_face, down_face, left_face, front_face, right_face, back_face, final_key, ciphertext
```

Figure 15: scramble_rot_key

The final step of the encryption process is the rot_cipher_key function will cipher do an XOR operation on our original key and our rot_key. The original key is repeated until its length is the same as the rot_key. The result given from this function is a very long key which lengths can vary based on the movements chosen by the scramble_rot_key function. Figure 16 is an example of a rot_cip_key.

```
123f284147522a3746515b25222c405a6121253046562d3746515d20202e435e662a284146563132365b5d5525383359
6d5a203537574e4d3e252f5d42583b2d155a584332323b464251393b365e405a123f2a424c553c3746585820345d465d
012b5845412c3e3545505c25332c405360212532465346534a3034585e5c3e3d3359665c20263554414022285d594e3d3159
615b203135544938202a5e59412d252a665f5f35355749463e3b2d5e41593b27175a5a4a26323b45405c252e345e4439
032b5b4243372f354650252e345e4652013f28414d563c233458545a3e2d3159635b203f37544047313e2f5d435d3b29
15595d4630323b45405d2e3b345d4559162b584740303b45425d202f465d4422062b58414d2c3a37455f5c2e202e405c
64212f3246544b26365b5d543e3a33596c5a2d3046514f2634585925222c40596c3d3e3246534f21345b585c3e29315a
67592f304651493836585f5a222c435d602a3e3246524932345b5b55352e435a622e2a414c5c3c354650593e202e405e
643a3e3046534b2136585a5d312c405361212f32465449323658555d332c405f6c2c3e3246573a37455a5c3e202c405f
103f284244273b45405e3e2f465c4022012b5b454c2c3435455a5f28365d405c013f284241502d35455e392f455b412d
17595e4b3024494c4f20202d455c4a3e175a513137544b41352a5d59463a315a665d2d2435544b40252a5d5e45
```

Figure 16: A result of the rot_cipher_key function

3.3 Movements file

The Movements file stores all possible movements in separate functions and has an array that acts as a legend to indicate which movement has been performed. On a 4x4 Rubik's cube, there are 72 different movements that can be performed, and these movements are stored in the movements array with a string representation of the movement as shown in Figure 17.

```
movements = [
    ["FA1", Fclock], ["FC1", Fanti], ["BA1", Bclock], ["BC1", Banti],["UA1", Uclock],["UC1", Uanti],["DA1", Dclock],["DC1", Danti],["RA1", Rclock],["RC1", Ranti],
    ["LA1", Lclock],["LC1", Lanti],["IFA1", innerfclock],["IFC1", innerfanti],["IBA1", innerbclock],["IBC1", innerbanti],["IUA1", inneruclock],["IUC1", inneruanti],
    ["IDA1", innerdclock],["IDC1", innerdanti],["IRA1", innerrclock],["IRC1", innerranti],["ILA1", innerlclock],["ILC1", innerlanti],["FWA1", Fwclock],["FWC1", Fwanti],
    ["BWA1", Bwclock],["BWC1", Bwanti],["UWA1", Uwclock],["UWC1", Uwanti],["DWA1", Dwclock],["DWC1", Dwanti],["RWA1", Rwclock],["RWC1", Rwanti],["LWA1", Lwclock],
    ["LWC1", Lwanti],["FA2", F2clock],["FC2", F2anti],["BA2", B2clock],["BC2", B2anti],["UA2", U2clock],["UC2", U2anti],["DA2", D2clock],["DC2", D2anti],["RA2", R2clock],["RC2", R2anti],
    ["LA2", L2clock],["LC2", L2anti],["IFA2", innerf2clock],["IFC2", innerf2anti],["IBA2", innerb2clock],["IBC2", innerb2anti],["IUA2", inneru2clock],["IUC2", inneru2anti],
    ["IDA2", innerd2clock],["IDC2", innerd2anti],["IRA2", innerr2clock],["IRC2", innerr2anti],["ILA2", innerl2clock],["ILC2", innerl2anti],["FWA2", Fw2clock],
    ["FWC2", Fw2anti],["BWA2", Bw2clock],["BWC2", Bw2anti],["UWA2", Uw2clock],["UWC2", Uw2anti],["DWA2", Dw2clock],["DWC2", Dw2anti],["RWA2", Rw2clock],["RWC2", Rw2anti],
    ["LWA2", Lw2clock],
    ["LWC2", Lw2anti]
]
```

Figure 17: Movements – Legend storing possible movements.

For the movements to be performed, each function requires that all the cube's face to be passed in as parameters. Some of these functions will be performed twice based on how much movement is required. All functions are programmed to move faces on the cube in a 90-degree angle either clockwise or anticlockwise. For movements requiring 180 degrees of movement, the same function will be executed twice. Figures 18 and 19 will demonstrate this.

```
5  def Fclock(up_face, down_face, left_face, front_face, right_face, back_face):
6
7      temp_up = [up_face[3][0], up_face[3][1], up_face[3][2], up_face[3][3]]
8      temp_right = [right_face[0][0], right_face[1][0], right_face[2][0], right_face[3][0]]
9      temp_down = [down_face[0][0], down_face[0][1], down_face[0][2], down_face[0][3]]
10     temp_left = [left_face[0][3], left_face[1][3], left_face[2][3], left_face[3][3]]
11
12     up_face[3][0], up_face[3][1], up_face[3][2], up_face[3][3] = temp_left[3], temp_left[2], temp_left[1], temp_left[0]
13     right_face[0][0], right_face[1][0], right_face[2][0], right_face[3][0] = temp_up[0], temp_up[1], temp_up[2], temp_up[3]
14     down_face[0][0], down_face[0][1], down_face[0][2], down_face[0][3] = temp_right[3], temp_right[2], temp_right[1], temp_right[0]
15     left_face[0][3], left_face[1][3], left_face[2][3], left_face[3][3] = temp_down[0], temp_down[1], temp_down[2], temp_down[3]
16
17     front_face = np.rot90(front_face, -1, axes=(0,1))
18     return up_face, down_face, left_face, front_face, right_face, back_face
```

Figure 18: Movements – F Clockwise Movement (90 Degrees).

```
def F2clock(up_face, down_face, left_face, front_face, right_face, back_face):
    up_face, down_face, left_face, front_face, right_face, back_face = Fclock(up_face, down_face, left_face, front_face, right_face, back_face)
    up_face, down_face, left_face, front_face, right_face, back_face = Fclock(up_face, down_face, left_face, front_face, right_face, back_face)
    return up_face, down_face, left_face, front_face, right_face, back_face
```

Figure 19: Movements – F Clockwise Movement performed twice (180 Degrees)

3.4 Conclusion

To conclude that the algorithm runs correctly through VSCode, press the play button in the top right as shown in Figure 20 on the main.py file.

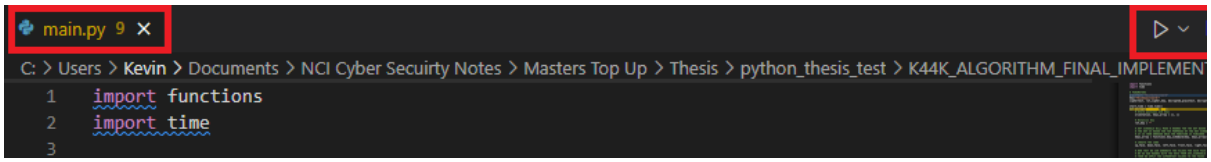


Figure 20: Executing the algorithms script

The results of the script can vary since the rot_cip_key variable can be created with different movements between each execution. Figures 22 and 23 shows two different rounds of encryption and decryption based off the same plaintext and key values which are also shown in Figure 21.

```
plaintext="Thistextwillwork"
key="Thiskeywillwork!"
```

Figure 21: Plaintext and key values

```
ENCRYPTION RESULTS
Ciphertext : 6757a4ff9dfed31b92d292f566f244a1b8b7692cf942ebfeacda2f344167b779cca90f68bd6d52b1404d0889bfd04b303c85921f0f11fe8e5
b1dfc1e3dfdd022f13a48c01083a4085fbb1aa6e977ce001b9dfcc020e2d0dd939d71d6adc73b31

ROT_CIP_KEY:
013f28425a5430332a5d5e402b252a13665a202628544f443c2f5e405f3b2d62655f58313c264b4e5a2539345d41587403295b4152292e36585a5b252e4358
1418295b475e273a465f5e3e345e4058631559584022303845 5d5e28365e4b596d032b5b445d2c2c365b5d5e3e2d315a13613a3e3059543d34585f5b3e3d33
5a12602a284153563f20285e5f332c435f16163f2842595d2c345b545e3e293359161d2e2a415d553033285d5f412d335916662c3e325a5d403b3e2f5d4357
3b3e606559503128574a47203e2f465d3b2f62665c5d3a2726484e58252e345e445d6315595c442d3238455c5c28345e445a6d032958425d30384651543934
5d455f74155a5c4229244b4f5e2a3b345d462274155a51313c24484359252a365d405d7403295847582c2c365b545a3138335913653d3e305a5041333e2d5e
4259302a106d3a2a425d5c2b20285e5b4f26342810605920312a574c432b3b2f455a47226515595c402d323a45585428345e422d7617595d4522373a465b5c
25312e405c161d2a2841535030222a5e544726272a106051203f28574c4e3c3b2f4558472d76175a5b4729244b4e5c392d465a273c60665b583f3c244b445b
392d455930281067513c305a5240223e2f5d425d363c6065595d353c244b415d282d455e472f76175a5047393238455a5439365e435c65175a504322303a46
585c2e345e4a5f6303295b4458292e365b5a5935383159176c3d3e325a5d48

ENCRYPTION ENDED

DECRYPTION STARTED
Plaintext (Uncovered by Ciphertext + Key): Thistextwillwork

ROT_Key (NOT HEXED)(Uncovered by ROT_CIP_KEY and Key): UWA111IDC127DWA222IUC163UC270IFC171BWC293IUC233UWA229LWA167RA135LA245BC
162RC123BA113IUA242DA192LWC276IUA212IBC125RWC21DC137IRA134BA283FWA23DC147BWA128UC282IFA27IFC260IDA136BA272DWA189LWC148IUA119BC
230IRC12IDC244ILC191IBC166BA157FWA250DC161LWA116UA188UC274UA251BA287FWC24IUA28BWA140IFA226UWA143IUA286FWA221UWC158DWA256BA19RC
169RWA278IFC141IBA254BWC255IDA153FWC218DC10FWC146IRC120IFA277IBA285IUC280IUA149ILC259UWC275FWC224BA295UA15UWA231LWA232UA26BC13
9UC179UWC152DWA114FWA264DA215DWC294RWA238UA117DC290IUC110BC184BWA273LWA265BWC268UWA181

DECRYPTION ENDED
```

Figure 22: Round 1 of algorithms result

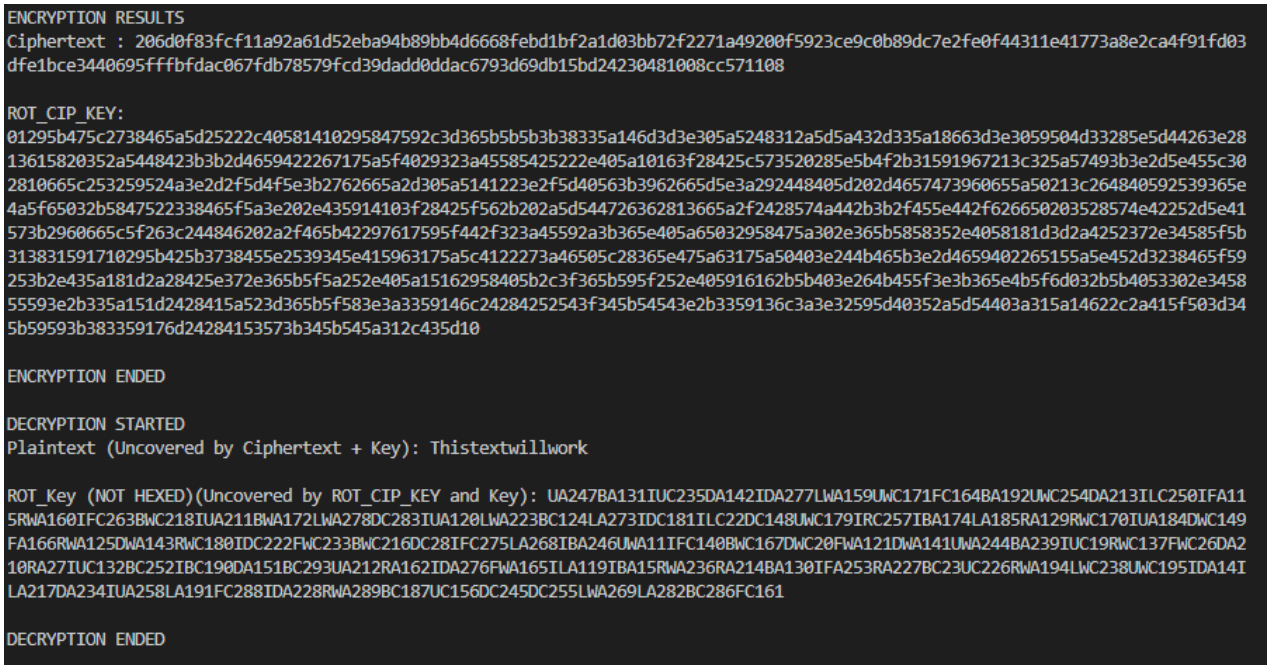


Figure 23: Round 2 of algorithms result

References

[1] https://www.mygreatlearning.com/. 2022. What is Numpy in Python | Python Numpy Tutorial. [online] Available at: <https://www.mygreatlearning.com/blog/python-numpy-tutorial/> [Accessed 20 June 2022].

[2] Loginradius.com. 2022. How does bitwise ^ (XOR) work? | LoginRadius Blog. [online] Available at: <https://www.loginradius.com/blog/engineering/how-does-bitwise-xor-work/> [Accessed 20 June 2022].

[3] Berent, A., 2022. AES (Advanced Encryption Standard) Simplified. 1st ed. [ebook] ABI Software Development, pp.7 - 8. Available at: <https://www.ime.usp.br/~rt/cranalysis/AESSimplified.pdf> [Accessed 21 June 2022].

[4] Educative: Interactive Courses for Software Developers. 2022. What is psutil.cpu_percent in Python?. [online] Available at: <https://www.educative.io/answers/what-is-psutilcpupercent-in-python> [Accessed 21 June 2022].