EEE3097S Progress Report 1

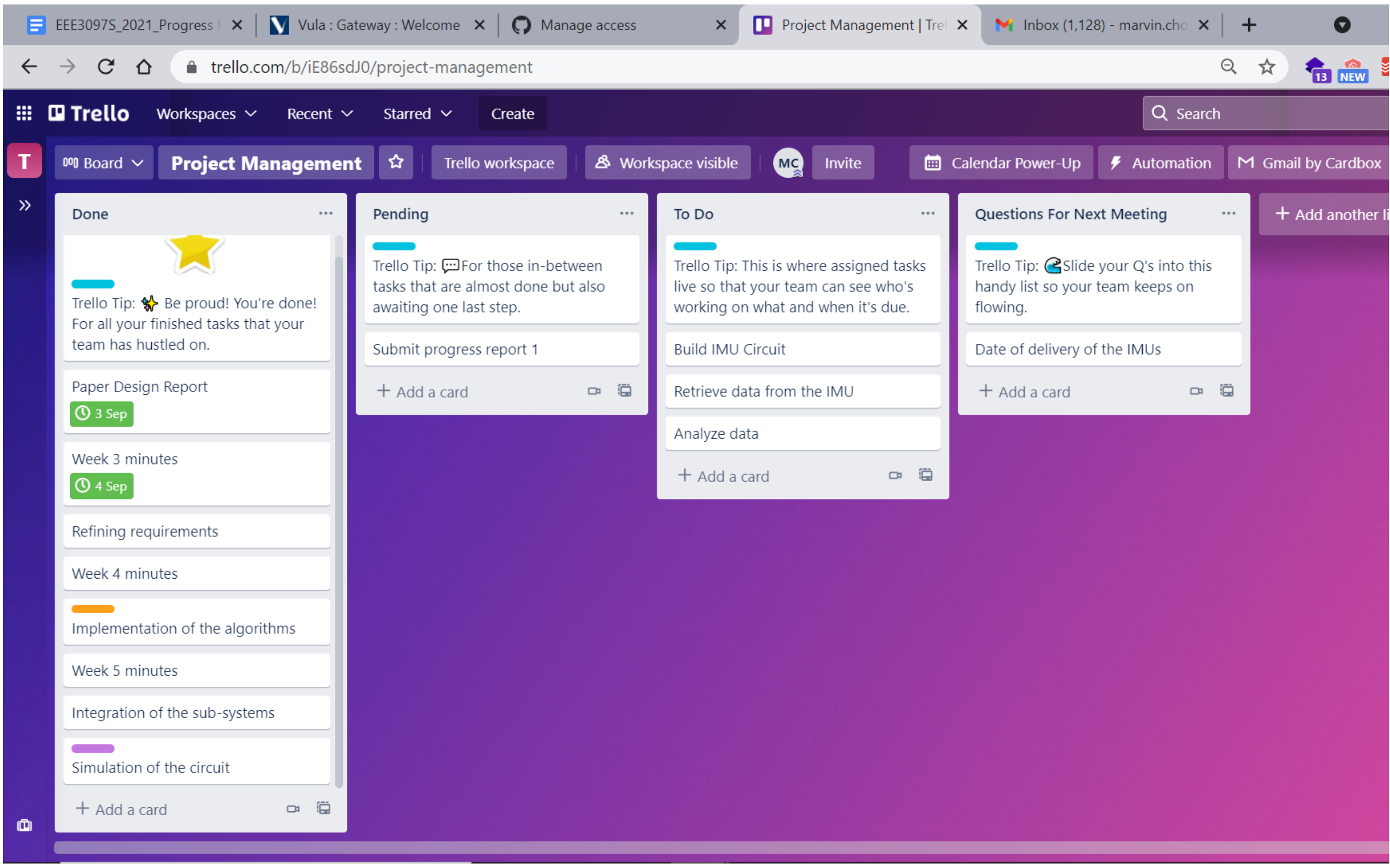
Department of

Prepared by : Lindani khuzwayo and Marvin Kangangi (Group 14)

Due :05 October 2021

Admin Documents

Marvin	Compression, Data Analysis
Lindani	Encryption, Data Analysis



The github page can be found [here]

The demo was done to the tutor (MKWZWA003) and the following remarks was noted:

- Data could be sent to the compression block and later transmitted to the encryption
- The compression and encryption worked as expected.
- The decryption and decompression worked as expected
- A comprehensive comparison was done to confirm that the data retrieval was successful

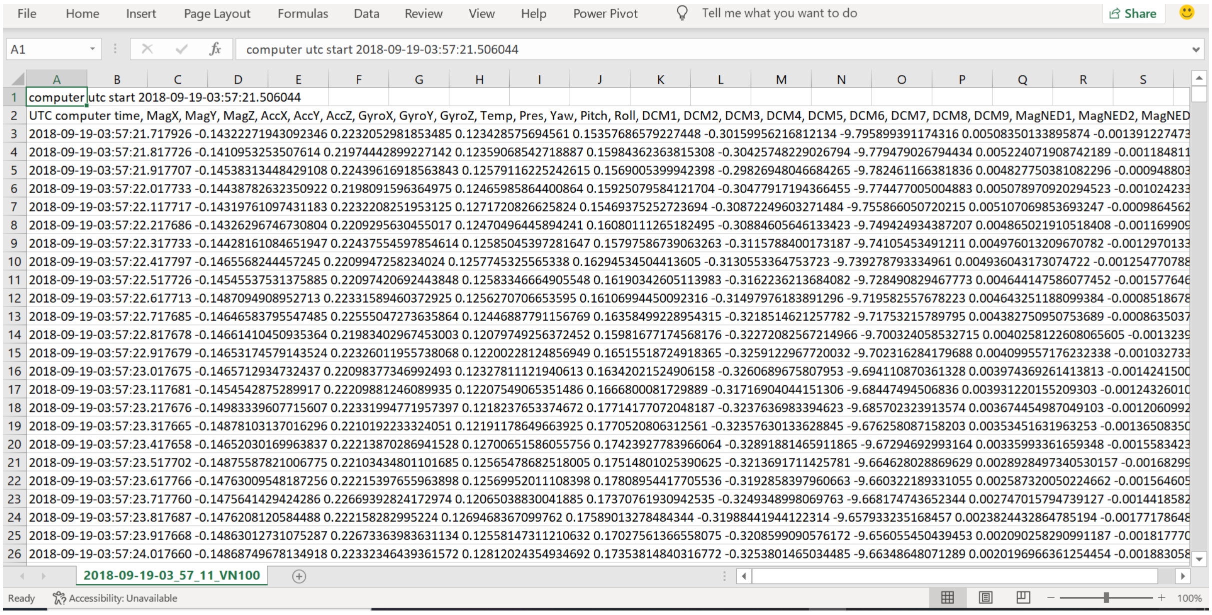
Timeline

- [x] ~~Project research and planning~~
- [x] ~~Paper design report~~
- [x] ~~Implementing the code on the Raspberry Pi~~
- [x] ~~Testing with sample data that resembles IMU data~~
- [ ] Testing with actual data from the IMU

We are on track to finish the project on the due date as stipulated.

Data

For now, we will use sample csv data shown below, that resembles the actual data from the IMU that will be used in later stages of the project. We decided to use this because it is sampled from previous data provided by an IMU. However, once an IMU is present for use, the data from the IMU will be used.



	A
1	computer utc start 2018-09-19-03:57:21.506044
2	UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM2, DCM3, DCM4, DCM5, DCM6, DCM7, DCM8, DCM9, MagNED1, MagNED2, MagNED
3	2018-09-19-03:57:21.717926 -0.14322271943092346 0.2232052981853485 0.12342857569456 1 0.15357686579227448 -0.30159956216812134 -9.795899391174316 0.0050835013389587 4 -0.00139122747 3
4	2018-09-19-03:57:21.817726 -0.1410953253507614 0.21974442899227142 0.12359068542718887 0.15984362363815308 -0.30425748229026794 -9.77947902679443 4 0.005224071908742189 -0.001184811
5	2018-09-19-03:57:21.917707 -0.14538313448429108 0.22439616918563843 0.12579116225242615 0.1569005399942398 -0.29826948046684265 -9.782461166381836 0.004827750381082296 -0.000948803
6	2018-09-19-03:57:22.017733 -0.14438782632350922 0.2198091596364975 0.12465985864400864 0.15925079584121704 -0.30477917194366455 -9.77447700500488 3 0.005078970920294523 -0.001024233
7	2018-09-19-03:57:22.117717 -0.14319761097431183 0.2232208251953125 0.1271720826625824 0.15469375252723694 -0.30872249603271484 -9.755866050720215 0.005107069853693247 -0.0009864562
8	2018-09-19-03:57:22.217686 -0.14326296746730804 0.2209295630455017 0.12470496445894241 0.16080111265182495 -0.30884605646133423 -9.749424934387207 0.004865021910518408 -0.001169909
9	2018-09-19-03:57:22.317733 -0.14428161084651947 0.22437554597854614 0.12585045397281647 0.15797586739063263 -0.3115788400173187 -9.74105453491211 0.00497601320967078 2 -0.0012970133
10	2018-09-19-03:57:22.417797 -0.1465568244457245 0.2209947258234024 0.1257745325565338 0.16294534504413605 -0.3130553364753723 -9.739278793334961 0.004936043173074722 -0.00125477078 8
11	2018-09-19-03:57:22.517726 -0.14545537531375885 0.22097420692443848 0.12583346664905548 0.16190342605113983 -0.3162236213684082 -9.72849082946777 3 0.004644147586077452 -0.001577646
12	2018-09-19-03:57:22.617713 -0.1487094908952713 0.22331589460372925 0.125627070665359 5 0.16106994450092316 -0.31497976183891296 -9.71958255767822 3 0.00464325118809938 4 -0.0008518678
13	2018-09-19-03:57:22.717685 -0.14646583795547485 0.22555047273635864 0.12446887791156769 0.16358499228954315 -0.3218514621257782 -9.71753215789795 0.004382750950753689 -0.0008635037
14	2018-09-19-03:57:22.817678 -0.14661410450935364 0.21983402967453003 0.12079749256372452 0.15981677174568176 -0.32272082567214966 -9.700324058532715 0.004025812260806560 5 -0.0013239
15	2018-09-19-03:57:22.917679 -0.14653174579143524 0.22326011955738068 0.12200228124856949 0.16515518724918365 -0.3259122967720032 -9.70231628417968 8 0.004099557176232338 -0.001032733
16	2018-09-19-03:57:23.017675 -0.146571293473243 7 0.22098377346992493 0.12327811121940613 0.16342021524906158 -0.3260689675807953 -9.694110870361328 0.003974369261413813 -0.0014241500
17	2018-09-19-03:57:23.117681 -0.145454287528991 7 0.22209881246089935 0.12207549065351486 0.1666800081729889 -0.31716904044151306 -9.68447494506836 0.003931220155209303 -0.0012432601 0
18	2018-09-19-03:57:23.217676 -0.14983339607715607 0.22331994771957397 0.1218237653374672 0.17714177072048187 -0.3237636983394623 -9.685702323913574 0.003674454987049103 -0.0012060992
19	2018-09-19-03:57:23.317665 -0.14878103137016296 0.2210192233324051 0.12191178649663925 0.1770520806312561 -0.32357630133628845 -9.676258087158203 0.003534516319632 53 -0.0013650835 0
20	2018-09-19-03:57:23.417658 -0.14652030169963837 0.22213870286941528 0.12700651586055756 0.17423927783966064 -0.32891881465911865 -9.67294692993164 0.003359933616593 48 -0.0015583423
21	2018-09-19-03:57:23.517702 -0.14875587821006775 0.22103434801101685 0.12565478682518005 0.17514801025390625 -0.3213691711425781 -9.664628028869629 0.002892849734053015 7 -0.00168299
22	2018-09-19-03:57:23.617766 -0.14763009548187256 0.22215397655963898 0.12569952011108398 0.17808954417705536 -0.3192858397960663 -9.660322189331055 0.002587320050224662 -0.001564605
23	2018-09-19-03:57:23.717760 -0.147564142942428 6 0.22669392824172974 0.12065038830041885 0.17370761930942535 -0.3249348998069763 -9.668174743652344 0.002747015794739127 -0.0014418582
24	2018-09-19-03:57:23.817687 -0.147620812058448 8 0.22158282995224 0.1269468367099762 0.17589013278484344 -0.31988441944122314 -9.657933235168457 0.002382443286478519 4 -0.00177178648
25	2018-09-19-03:57:23.917668 -0.14863012731075287 0.22673363983631134 0.12558147311210632 0.17027561366558075 -0.3208599090576172 -9.656055450439453 0.002090258290991187 -0.00181777 0
26	2018-09-19-03:57:24.017660 -0.14868749678134918 0.22332346439361572 0.12812024354934692 0.17353814840316772 -0.3253801465034485 -9.66348648071289 0.002019696636125445 4 -0.001883058

The table below from the paper design specifies the expected data from the IMU which corresponds to the special axial components specified below.This data is the compressed and encrypted from one point for example locally (PC) which prepares for remote transfer of the files which are already secured, thus for the purpose of transmitting. Compression and encryption won't be necessary if there was no remote transmission of data.

Table 1

X-axis acceleration	Signed 16-bit integer
Y-axis acceleration	Signed 16-bit integer
Z-axis acceleration	Signed 16-bit integer
X-axis angular velocity	Signed 16-bit integer
Y-axis angular velocity	Signed 16-bit integer
Z-axis angular velocity	Signed 16-bit integer

Experiment Setup

<u>Entire System:</u>

The latency of the system was put into test. In order to implement this, the native python **time** module was put into use. A timer was set before the system began and stopped right after. It should be noted that the timer was set only for the critical sections (compression and encryption) and not the trivial functions such as printing and reading files. Results were recorded and a graph plotted for a clearer representation.

A test was carried out to test the output of the system. A csv file was fed into the system and the output was examined for correctness. It was expected that an ".compressed.txt.enc" file would be present in the output after running the system. A python script was implemented to do this. The input being the name of the original file pi@raspberrypi: ~/Documents/data

```
    I A   does_output_exist.py  (python)   def check_file(file_n
import sys
from pathlib import Path

def check_file(file_name):
    my_file = Path(sys.argv[1][:-4] + "-compressed.txt.enc")
#     print(my_file)
    if my_file.is_file():
        print("True")
    else:
        print("False")


if __name__ == "__main__":
    check_file(sys.argv[1])
```

Figure 1.1

The final test carried out was to ensure that there is proper communication between the compression and encryption. This was done by making sure the encryption system had input from the compression system at all times. Socket programming was used for the transmission.

Compression

For compression, a couple of tests were carried out to validate the subsystem. For the purposes of this project, the input was in csv format from the IMU. The output was both in csv and text. The csv was to provide a good comparison between it and the original while the text file was the preferred input of the encryption system.

The first test that was carried out was to check whether the compression subsystem actually produces a compressed file. To do this, the data csv file was fed into the subsystem. The subsystem will then create a csv and text file with a “compressed” suffix. A python script (does_compressed_file_exist.py) was written to implement this. If there is a compressed version of the file, it will return true and false otherwise. It’s expected that a file is present after the compression algorithm is run. The name of the file being tested with is “2018-09-19-06_53_21_VN100.csv” pi@raspberrypi: ~/Documents/data

```
  I A  does_compressed_file_exist.py (Modified) (python)  de  Row 6    Col 1
import sys
from pathlib import Path

def check_file(file_name):

    my_file = Path(sys.argv[1][:-4] + "-compressed.csv")
    print(my_file)
    if my_file.is_file():
        print("True")
    else:
        print("False")


if __name__ == "__main__":
    check_file(sys.argv[1])
```

Figure 2.1

The second test that was carried out was to ensure that the compressed file was smaller in size as compared to the original file. To effectively carry this out, native python functions in the test_compress.py file were used to compare the size of the two files in terms of bytes and a comparison was carried out. It's expected that the compressed file will be significantly smaller.

pi@raspberrypi: ~/Documents/data

```
  I A  test_compress.py (Modified) (python)  print("File siz  Row 6    Col 1
import os
import sys

file_size = os.path.getsize(sys.argv[2])
print("File size of " + sys.argv[2] + " is :", file_size, "bytes")

file_size2 = os.path.getsize(sys.argv[1])
print("File size of " + sys.argv[1] + " is :", file_size2, "bytes")
```

Figure 2.2

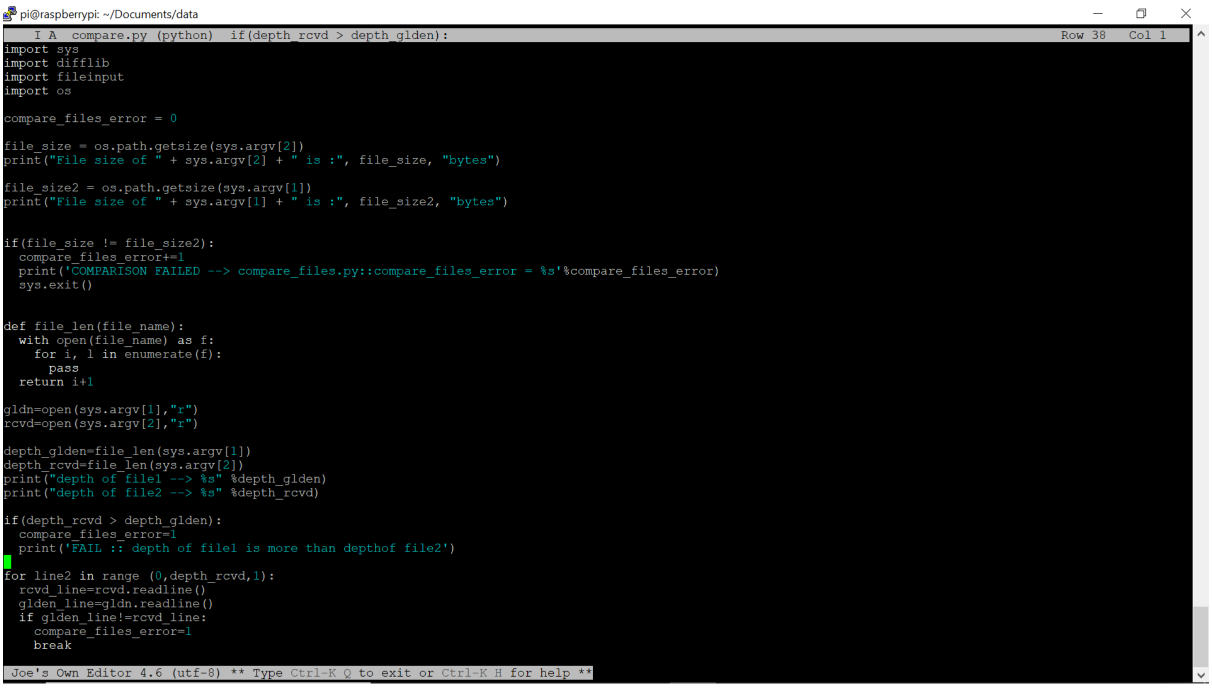
After getting the file sizes of both the original data and the compressed file, the compression ratio of the algorithm was calculated. The following formula was used:

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

Figure 2.3

This was tested for different sizes and types of data and the results tabulated. It's expected that the compression ratio should be similar or deviate by a small factor. It's also expected that the compression ratio will be small, within the range of 1 to 4 due to the lossless nature of the algorithm.

Another test was carried out to make sure that the decompressed file was the same as the original file thus ensuring no loss of data.To do this, three different tests were carried out in the same python script (compare.py). The first test was done to determine whether the two files were of the same size. The second test that was done was to determine whether the files had the same length and finally content. It iterates through the contents of the files and compares the two. If there is a discrepancy in any of the three tests, the script returns an error and halts operation of the whole experiment. With these three parameters, we believed that the lossless nature of the algorithm would be adequately put to test. It's expected that there will be no loss in the data since the gzip algorithm used is lossless.The code is shown in Figure 2.4.



```
pi@raspberrypi: ~/Documents/data
    I A   compare.py  (python)   if(depth_rcvd > depth_glden):                Row 38    Col 1
import sys
import difflib
import fileinput
import os

compare_files_error = 0

file_size = os.path.getsize(sys.argv[2])
print("File size of " + sys.argv[2] + " is :", file_size, "bytes")

file_size2 = os.path.getsize(sys.argv[1])
print("File size of " + sys.argv[1] + " is :", file_size2, "bytes")


if(file_size != file_size2):
  compare_files_error+=1
  print('COMPARISON FAILED --> compare_files.py::compare_files_error = %s'%compare_files_error)
  sys.exit()


def file_len(file_name):
  with open(file_name) as f:
    for i, l in enumerate(f):
      pass
  return i+1

gldn=open(sys.argv[1],"r")
rcvd=open(sys.argv[2],"r")

depth_glden=file_len(sys.argv[1])
depth_rcvd=file_len(sys.argv[2])
print("depth of file1 --> %s" %depth_glden)
print("depth of file2 --> %s" %depth_rcvd)

if(depth_rcvd > depth_glden):
  compare_files_error=1
  print('FAIL :: depth of file1 is more than depthof file2')

for line2 in range (0,depth_rcvd,1):
  rcvd_line=rcvd.readline()
  glden_line=gldn.readline()
  if glden_line!=rcvd_line:
    compare_files_error=1
    break
 Joe's Own Editor 4.6 (utf-8) ** Type Ctrl-K Q to exit or Ctrl-K H for help **
```

Figure 2.4

A test was finally carried to test the speed of the subsystem. This was done using the native python **time** module. A timer was set before the function call and stopped after the function. This gave us the execution time of the function. The time was tested on various sets of data of different sizes and formats in order to draw accurate conclusions. Appropriate graphs were drawn to provide a visual. It should be noted that the timer was only set for the critical parts of the code i.e the compression and not other trivial functions like printing and opening the files.

Encryption and Decryption :

Below is a set of steps taken to check whether a file was successfully encrypted/decrypted or not. Data used is this section as input a text file and output text.enc file for the encryption part. Lastly input for decryption is a text.enc file and outputs the original txt file :

1. First volition was checking whether the encryption mechanism is reversible or not which is a very important part of this project. That is because failure of reversing the encrypted file means the origins data is lost or not accessible

2. Secondly, making sure that no one else can access the original data without the decruping script with the correct key ,as that will defeat the purpose of encryption.This is successfully done by changing using a random key on the decryption script. Also making sure that the key is not known. This is completely dependent on

the type of key used in this case a 258 bit key is used which is generally difficult to figure out.

3. Thirdly, looking at the entropy of the file. If the entropy is high, then it’s likely encrypted. You can use tools like binwalk to determine the entropy. A consistent, high entropy indicates that the file is likely encrypted.To investigate files for hidden threats, you can look at file entropy values.File entropy measures the randomness of the data in a file and is used to determine whether a file contains hidden data or suspicious scripts. The scale of randomness is from 0, not random, to 8, totally random, such as an encrypted file. The more a unit can be compressed, the lower the entropy value; the less a unit can be compressed, the higher the entropy value.

   Just examining the file using tools like “strings” will also give you a clue, but won’t be as definitive as binwalk. If you find actual readable strings in the file, it’s not encrypted. However, some forms of compression will often look like encryption. If there’s a readable header in the file, then it’s likely compressed and not encrypted.

- **Results**

Entire System:

1. **Speed of the system**

The tests explained earlier were implemented and the following results were obtained.

	Data size (Bytes)	Speed(s)
1	5709	0.027378985
2	56890	0.15123934
3	623134	0.6729736324
4	1247263	1.27062843
5	6240886	6.5621285

Table 2.1

Using the data above, the graph below was plotted:

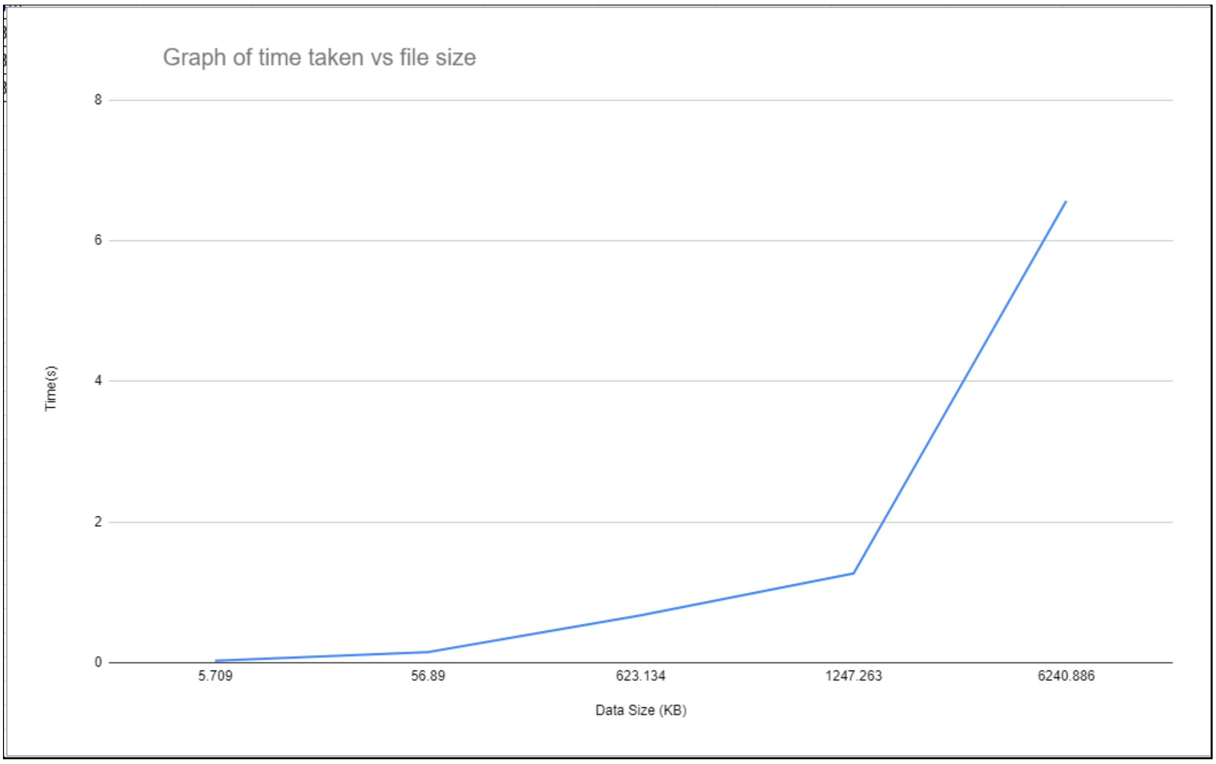


Figure 3.1

It was seen that the time increased exponentially as the file size increased.

2. **Desired output test**

   Using the python script, a check was done to determine w

```
pi@raspberrypi: ~/Documents/data
pi@raspberrypi:~/Documents/data $ python3 does_output_exist.py data.csv
True
```

This confirmed that indeed the required output was produced by the system. The validity of the output was tested in the later stages of the tests.

3. **Data transmission test**

The fact that the system ran and produced an output indicated that it indeed had an input and therefore there was proper communication between the two systems

Compression:

1. **Compressed file is produced**

After running the script, the following result was obtained:

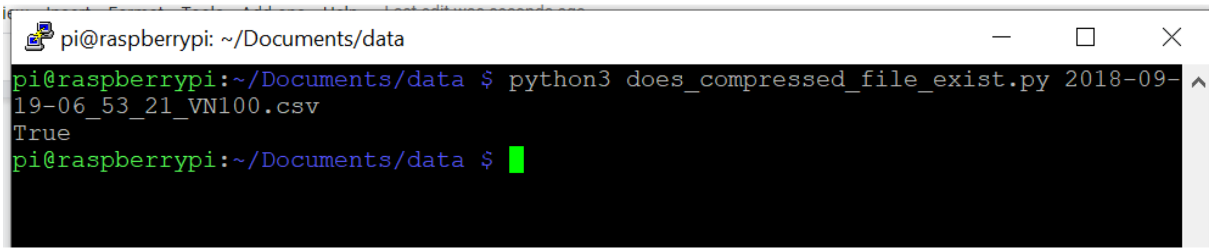


Figure 3.2

To test for when there's no file, the suffix "-error" was added to the file name in the command line arguments.

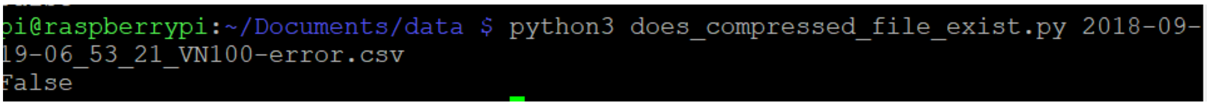


Figure 3.3

This shows that the compression algorithm did indeed produce a file. This file will be tested further to ensure its validity.

2. The size of the compressed file

With the script shown in Figure 2.2, the following result was obtained.

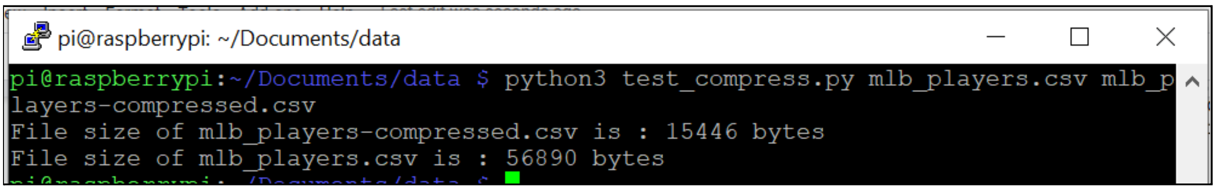


Figure 3.4

The size of the original file was 56.890 KB and the compressed file is 15.446 KB. This led to the conclusion that the compression algorithm indeed compressed the file as required.

3. Compression ratio

The formula shown below was used to determine the compression ratio of different data sizes and results were tabulated.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

	Data size	Compression ratio
1	5709	3.0859459
2	56890	3.6831542
3	623134	3.160918
4	1247263	3.175062
5	6240886	3.187314

Table 3.1

The compression ratio of the algorithm with different sets and sizes of data was **3.2584**.

The compression ratio is not high due the algorithm’s lossless nature. The compression ratio of lossless algorithms range from 1:1 to 4:1. The higher the compression ratio, the smaller the file size. However, this will take significantly more resources to decompress back to the original file. Thus a compression ratio of 3.25 provides a good balance between size of compressed file and ease of decompression.

4. Loss of data

The algorithm used is a lossless algorithm and therefore it was expected that the decompressed file would be the same in size and contents. Running the tests explained earlier, we obtained the following results.(Name of the original file is data.csv and the name of the decompressed file is data-v2.csv)

```
pi@raspberrypi:~/Documents/data $ python3 compare.py data.csv data-v2.csv
File size of data-v2.csv is : 9071107 bytes
File size of data.csv is : 9071107 bytes
depth of file1 --> 14940
depth of file2 --> 14940
COMPARISION SUCCESSFUL --> compare_files.py::compare_files_error = 0
```

Figure 3.5

It was noted that the files had the same size, length and content and therefore 0 errors present.

In order to fully test it, a bit was added to data-v2.csv to check whether the script would return an error. This is what was observed.

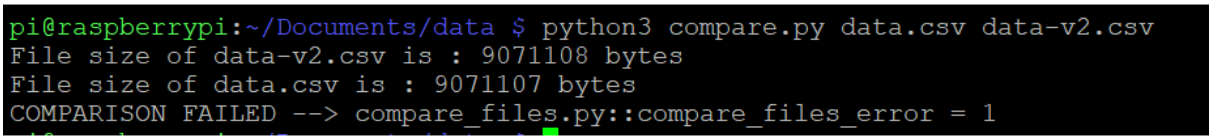


Figure 3.6

Upon adding a bit, the script returned an error. This further confirmed that the original file and the compressed file were equal and there was no data loss

5. **Time taken to compress data**

The algorithm was tested using different data sizes and the results were recorded in Table 1 below and a corresponding graph was plotted.

	File size (bytes)	**Time taken(s)**
1	5709	0.002556085
2	56890	0.12375044
3	623134	0.614487886
4	1247263	1.1698408
5	6240886	6.1944408

Table 3.2

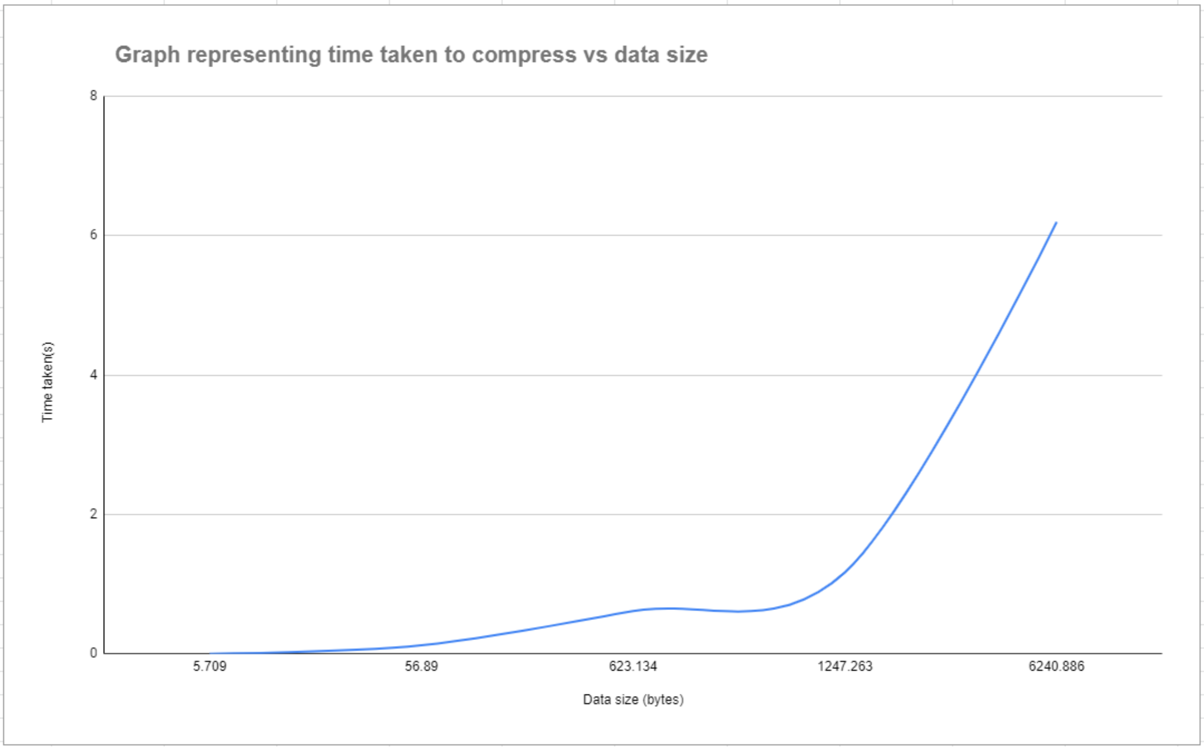


Figure 3.7

From the results above, we can see that the time taken increases exponentially and not linearly as the data size grows thus the smaller the data size, the faster the compression

Encryption:

1.**Recovery test**

The results are analyzed based on the implementation discussed above.figure 2.1 below is shows the original data before it is encrypted

```
computer utc start 2018-09-19-04:22:31.529971
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
2018-09-19-04:22:31.793573 -0.34008175134658813 0.35862863063812256 0.30060529708862305 0.23097454011440277 -0.3260
2018-09-19-04:22:31.893383 -0.342180460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
2018-09-19-04:22:31.993358 -0.34227073192596436 0.35868507623672485 0.3042473793029785 0.23394353687763214 -0.32825
2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331705
2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32249
2018-09-19-04:22:32.693342 -0.34121447801589966 0.3552418649196625 0.3081103265285492 0.2025754451751709 -0.3199371
2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.18888604640960693 -0.31485
2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
2018-09-19-04:22:33.093386 -0.3422207236289978 0.36097633838653564 0.30421796441078186 0.18590101599693298 -0.31363
2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551810681819916 0.3032151758670807 0.18188047409057617 -0.3065259
2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.1633971929550171 -0.297162
2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
2018-09-19-04:22:33.793317 -0.3401374816894531 0.3540785312652588 0.30818718671798706 0.15837541222572327 -0.294530
2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.15909089148044586 -0.2921
2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.157296106219291 7 -0.283139
2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
2018-09-19-04:22:34.493330 -0.3378465175628662 0.3586127758026123 0.3057440221309662 0.1454452574253082 3 -0.2623203
```

Figure 4.1:

The file shown in figure 2.1 is encrypted through the script shown in figure 2.2 below. The resulting encrypted file is shown in figure 2.3

```
import sys
from encryptor import Encryptor

#key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\

key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\xcb\xc4\x94\x9d(\x9e'
encryptor1 = Encryptor(key)

encryptor1.encrypt_file(sys.argv[1])
```

Figure 4.2 : encrypt.py script

Figure 4.3: content of the encrypted file

For the purpose of this experiment, checking if the encryption mechanism is reversible or not the very same file shown above figure 2.3 is run through the decryption script namely decrypt.py shown in figure 2.4 the results from that process is shown in figure 2.5 this does not only prove successful encryption but also decryption.

IMU_prac-master > decrypt.py > ...

```python
import sys
from encryptor import Encryptor

#key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\

key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\xcb\xc4\x94\x9d(\x9e'
encryptor1 = Encryptor(key)

encryptor1.decrypt_file(sys.argv[1])
```

figure 4.4:decrypt.py

```
computer utc start 2018-09-19-04:22:31.529971
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
2018-09-19-04:22:31.793573 -0.34008175134658813 0.35862863063812256 0.30060529708862305 0.23097454011440277 -0.3260
2018-09-19-04:22:31.893383 -0.342180460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
2018-09-19-04:22:31.993358 -0.34227073192596436 0.35868507623672485 0.3042473793029785 0.23394353687763214 -0.32825
2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331705
2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32249
2018-09-19-04:22:32.693342 -0.34121447801589966 0.3552418649196625 0.3081103265285492 0.2025754451751709 -0.3199371
2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.18888604640960693 -0.31485
2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
2018-09-19-04:22:33.093386 -0.342220723628997 8 0.36097633838653564 0.30421796441078186 0.1859010159969329 8 -0.31363
2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551810681819916 0.3032151758670807 0.18188047409057617 -0.3065259
2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.163397192955017 1 -0.297162
2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
2018-09-19-04:22:33.793317 -0.3401374816894531 0.3540785312652588 0.30818718671798706 0.15837541222572327 -0.294530
2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.15909089148044586 -0.2921
2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.1572961062192917 -0.283139
2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
2018-09-19-04:22:34.493330 -0.3378465175628662 0.3586127758026123 0.3057440221309662 0.1454452574253082 3 -0.2623203
```

figure 4.5

2. **Key integrity testing**

The outcome from changing the key when decrypting clearly shows that one cannot recover the original data.Above the figure 2.1 was encrypted with the key show in figure 2.6 and later decrypted with a different 16it key shown in figure 2.8 and the outcome is represented by figure 2.9 This proves and validates security as promised by the AES algorithm

```
key = b'0123456789987654'
enc = Encryptor(key)
```

4.6: new encryption 16 bit key

4.7:result from figure 2.6

IMU_prac-master > decrypt.py > ...

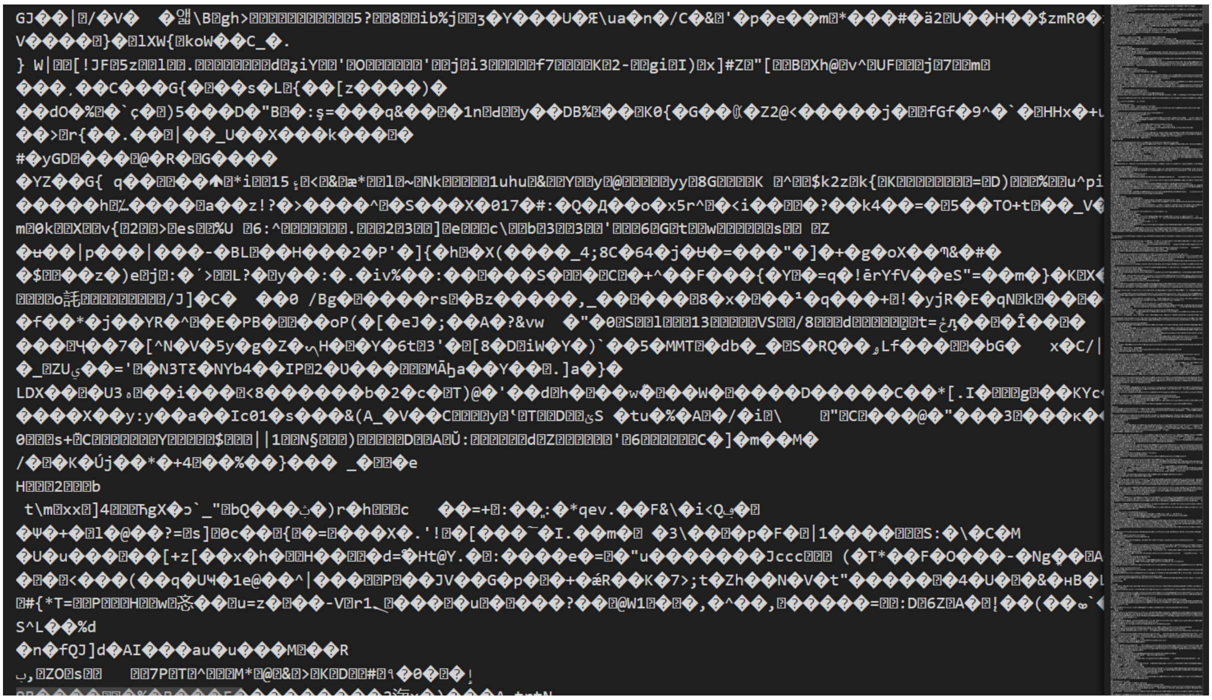
```
import sys
from encryptor import Encryptor

#key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\

key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\xcb\xc4\x94\x9d(\x9e'
encryptor1 = Encryptor(key)

encryptor1.decrypt_file(sys.argv[1])
```

4.8:decrypting key(original key for the projet)



4.9:result from figure 2.8

3. <u>Entropy Test</u>

Below are results from Entropy and Randomness Online Tester of the original file 2018-09-19-03_57_11_VN100.CSV and the encrypted file 2018-09-19-03_57_11_VN100.CSV.ENC

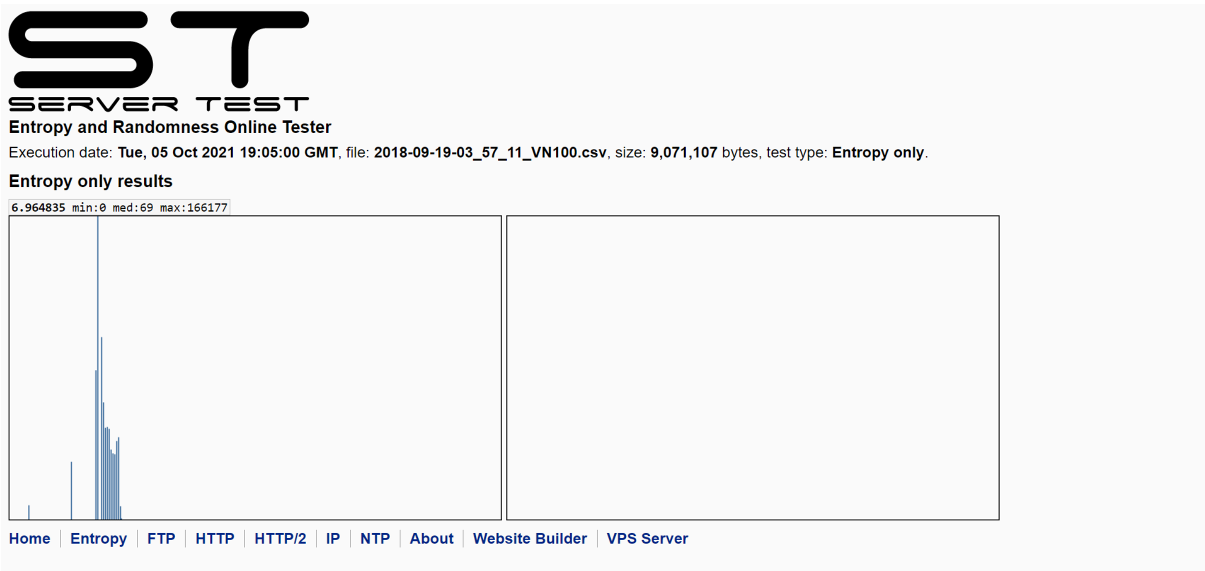


Figure 5.1: file2018-09-19-03_57_11_VN100.CSV

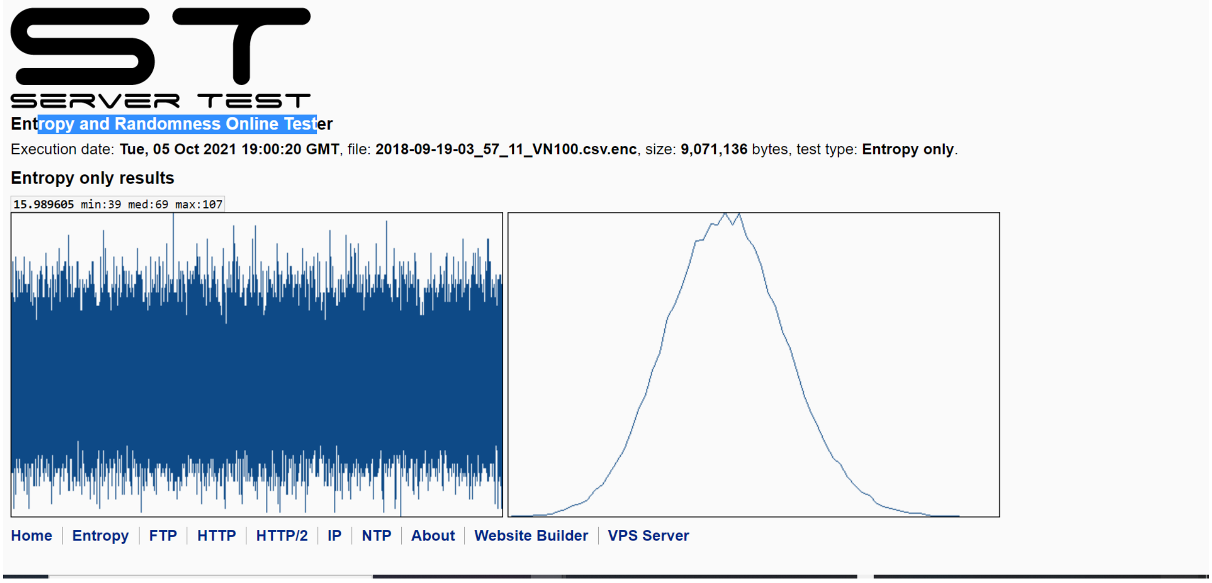


Figure 5.2:file 2018-09-19-03_57_11_VN100.CSV.ENC

- **ATPs**

Entire System:

AT001	**Configuration of the Slave**
Evaluation type	Hardware
Target	I2C connection
Test protocol	• Connect the IMU to the master, read and write from IMU registers • Additional self test
Pass condition	• Start and stop condition generation •
Fail condition	No response from the IMU
Test result	Awaiting IMU to test.

Compression:

AT002	**Compressed file size test**
Evaluation type	Software
Target	Compression subsystem
Test protocol	• Feed the sample data into the subsystem • Determine and compare the size of the output with the original
Pass condition	The output file's size is smaller than the input
Fail condition	The output file's size is equal or bigger than the original file.
Test result	Pass

AT003	Data loss test
Evaluation type	Software
Target	Compression subsystem
Test protocol	• Feed the sample data into the subsystem • Extract the output of the subsystem and decompress it. • Compare this with the original file
Pass condition	The decompressed file is the same as the original file
Fail condition	The decompressed file differs from the original file
Test result	Pass

Added ATPs

AT005	File produced test
Evaluation type	Software
Target	Compression subsystem
Test protocol	• Feed the sample data into the subsystem • Determine whether an output is produced by the system
Pass condition	A file is produced by the subsystem
Fail condition	No file is produced
Test result	Pass

All the previous ATPs were met and therefore the specifications do not change.

Encryption and Decryption :

Results of the experiments done above are compared with our ATPs in this section.

AT004	Encryption/decryption algorithm correctness and successful execution
Evaluation type	Software
Target	Encryption algorithm
Test protocol	• Simulation of the encryption and decryption process using the open source code here
Pass condition	• Successfully encrypt the given data • Successfully decryption of the given encrypted file with a key back to the original data
Fail condition	• Non recovery of the original data
Test results	pass

AT004 was successfully executed and it follows that the it is met

Added ATPs

AT006	Entropy and Randomness Test
Evaluation type	Software
Target	Encrypted files
Test protocol	• Entropy formula measures the equiprobability regardless of real unpredictability. • Tool can be found here
Pass condition	• High Entropy score • Dense Entropy graph is more desirable for a pass condition • See figure 3.1
Fail condition	• Low entropy score • Non dense Entropy graph
Test results	pass

AT007	Encryption/decryption algorithm secured key

Evaluation type	Software
Target	Encryption and decryption algorithm
Test protocol	• Simulation of the encryption and decryption process using a different key for these stages
Pass condition	• Non recovery of the original data • See under experiment results 2
Fail condition	• Successfully encrypt the given data • Successfully decryption of the given encrypted file with a key back to the original data
Tes results	Pass

All the previous ATPs were met and therefore the specifications do not change.