EEE3097S Final Report Submission

Samuel Goodson (GDSSAM001)

Tumza Seeco (SCXITU001)

Admin Documents:

Table of contributions:

Samuel Goodson	Encryption and Decryption. Responsible for coding the entire Encryption and Decryption algorithm using AES encryption from the PyCryptoDome package. Both systems work as intended.
Tumza Seeco	Compression and Decompression. Responsible for implementing Gzip to compress and decompress the files.
Samuel Goodson	IMU implementation. Responsible for setting up the sense Hat on the Pi and using python to generate IMU data for the testing
Samuel Goodson	Asana Board
Samuel Goodson	Experimentation and Results
Samuel Goodson	IMU simulation
Tumza Seeco	ATPs
Samuel Goodson	Demonstration Video
Samuel Goodson	Requirements Analysis
Samuel Goodson	Validation using IMU simulation
Samuel Goodson	Validation using a different IMU
Samuel Goodson	Conclusion

Github repo: https://github.com/SamuelGitson/EEE3097S_GDSSAM001_SCXITU002.git

Timeline from the beginning:

✓ Mondays/Tuesdays/Fridays Had multiple meetings per week to discuss objectives and how we were going to split the project between us

✓ 21/08/2021: Decided on project management tool (Asana)

✓ 27/08/2021: Reviewed possible encryption and compression algorithms. Delegated each subsection to one another

✓ 03/09/2021: simulated IMU data in Matlab and successfully sent it to a .txt file and compressed it using Gzip in python.

✓ 04/09/2021: Met on MS Teams to start writing up the paper design for submission.

✓ 05/09/2021: Finished up paper design and prepared it for submission

✓ Mondays/Fridays we had our meetings to discuss objectives and how we were going to split the project between us

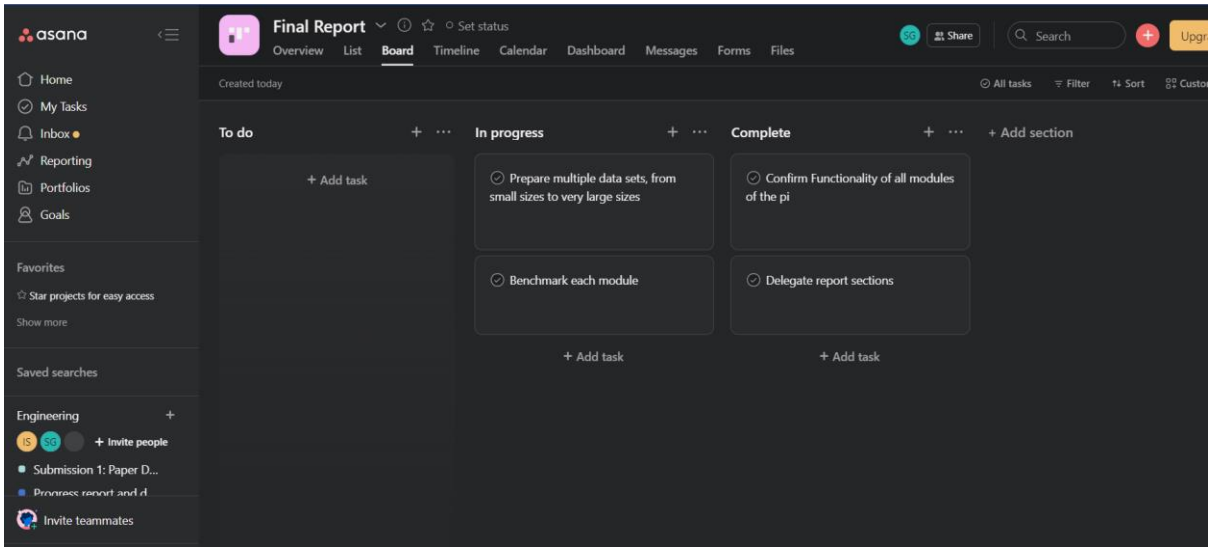
✓ 26/10/2021: Ported python code onto the Pi and made the necessary changes to it. Added IMC data simulation for the sense Hat and sent that data to a nano file that would be the target for compression and encryption.

✓ 27/10/2021: Did the required testing and finalised second progress report. We have run behind schedule by three days on this progress report. However, we aim to finish the final report comfortably before the final report deadline.

✓ Mondays/Fridays we had our meetings to discuss objectives and how we were going to split the project between us

✓ 26/10/2021: Ported python code onto the Pi and made the necessary changes to it. Added IMC data simulation for the sense Hat and sent that data to a nano file that would be the target for compression and encryption.

✓ 27/10/2021: Did the required testing and finalised second progress report. We have run behind schedule by three days on this report. However, we aim to finish the final report comfortably before the final report deadline.



Requirements Analysis:

For this project we were required to be able to get IMU data and then compress and encrypt it. Once successfully compressed and encrypted, we needed to then be able to decompress and decrypt the data successfully, meaning we should have been able to read the original data after decompressing and decrypting it. These algorithms needed to be function on a Pi.

The requirement needing IMU data meant that we were required to attach a sense Hat(B) to our Pi and then use python code to give us data from the IMU on in the sense Hat(B). The easiest ICM to do this was very similar to the given shark buoy ICM, with only slight differences. The compression algorithm needed to meet a certain threshold, so it was required of us to code a compression system that met this threshold. Our encryption method needed to be efficient and still allow for the code to be decryptable without losing any data.

For encryption we were faced with three main options. These were AES, RSA, and SHA-1. AES seemed like least complicated out of the three, so we went with this one. We did some testing with

various compression libraries. Gzip worked better for out file systems than Zlib and LZMA did. So, we chose Gzip as our compression library.

For the specifications:

To get IMU data as close as possible to the shark buoy, we used a sense Hat(b) and downloaded python code for an ICM-20948, which is similar but not the same as the ICM-20649 on the shark buoy. The main differences being that it can't function properly under as much stress as the shark buoy ICM and in high mobility situations, it may be less accurate. However, for this project is it more than capable of the stress that we plan on putting it through. As mentioned above, we implemented AES encryption, which operates by turning plane text into cypher text using a key. We planned to use the smallest key size in an effort to maximize efficiency over security.

To compress the files by the required amount, we used GZIP. This library is quite easy to use and is able to effectively compress files of any size.

IMU samples need to be able to run for at least 100, 200, 300, and 500 samples

Total compression must be under 1 second.

AT001	**Encryption Test**
Evaluation type	Software Unit Test
Target	AES Encryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The encryption algorithm must be used to generate a key and use it to encrypt the data.
Pass condition	• The 128-bit key is generated and stored. • Data is encrypted. • Encrypted file is generated
Fail condition	• A key is not generated • Output file data looks the same as input data; meaning that the data is not encrypted. • File is corrupted • There is no output file. • Key size is incorrect.

AT002	**Encryption Test**
Evaluation type	Software Unit Test
Target	AES Decryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The decryption algorithm must be used to decrypt the data.
Pass condition	• The generated Key is stored safely and is used to decrypt data. • The output file is the same as the original file after compression.
Fail condition	• Output file data looks different to the original 'unencrypted' data • File is corrupted • There is no output file • The key fails to decrypt the file.

AT003	**Compression Test**
Evaluation type	Software Unit Test
Target	Compression Algorithm Subsystem
Test Protocol	Gzip Compression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new compressed file being generated.
Pass condition	• The text-file is compressed, i.e., the file size has been reduced significantly (45-55%)
Fail condition	• There is no output file. • The file output is not compressed, i.e., the file size is the same as the original file.

AT004	**Decompression Test**
Evaluation type	Software Unit Test
Target	Decompression Algorithm Subsystem
Test Protocol	Gzip Decompression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new decompressed file being generated.
Pass condition	• The text-file is decompressed, i.e., the file size has been restored to the original.
Fail condition	• There is no output file. • The file output is different to the initial file.

AT005	Specification Test
Evaluation type	Parameters
Target	Hardware components
Test Protocol	Evaluate ICM-20948 specifications from the datasheet and check if these meet the ones requirements for the system that we are building.
Pass condition	• The specifications meet the desired requirements
Fail condition	• Specifications do not meet the desired requirements.

AT007	System Robustness
Evaluation type	Software
Target	System and Subsystems
Test Protocol	Connect the ICM-20948 to the Raspberry Pi and take measurements and store them in a nano file. Moreover, run the entire system (AES encryption and Gzip algorithms), i.e., producing the compressed and Encrypted files. Run this process 100, 200, 300, and 500 times.
Pass condition	• The Raspberry Pi can successfully complete the consecutive cycles.
Fail condition	• Failure to run the cycles. • Failure to encrypt the data • Failure to compress the data

Paper Design:

For this report, we cover our chosen subsystems and sub-subsystems, as well as their proposed requirements and specifications. We also aim to talk to about possible shortfalls and bottlenecks our design might run into and how to avoid these. This report aims to make our goals and methods clear as to how we plan to achieve our design. The report will also cover our individual contributions and give images of how we have been using our project management tool, Asana.

Requirement analysis:

Our design is required to be able to take IMU data from the sharc-buoy using a motion senor, and then compress and encrypt it before sending it. There are other technical requirements involving the compression and encryption.

Compression:

Requirements for compression:

- Compression needs to allow for enough fourier coefficients to be extracted by the oceanographers (25% of fourier coefficients).
- The algorithm needs to be efficient as to reduce power consumption

Available compression and encryption algorithms:

For compression, we have decided to use GZIP. It is very simple to make use of and is effective. Other available options are ZLIB and LZMA. These options can be looked at later if it is found that Gzip is unable to compress the data to the required amount. For encryption, we have decided to use the AES encryption algorithm.

Encryption:

Requirements for encryption:

- Encryption method needs to be safe and work over networks that are unsecure

- Encryption needs to accurate, and data cannot be lost in the encryption/decryption process

Available encryption methods:

	AES	RSA	SHA-1
Description	This type of encryption works by converting plain text into cipher text using a symmetric key encryption technique	This type of encryption uses asymmetric cryptographic algorithm that uses two distinct keys; one is public (used to encrypt text) and the other is private (used to decrypt text)	This is a cryptographic hash function which uses the input to create a 20 byte hash value (message digest). The algorithm results in an output of blocks of 16 words; this output will be used instead of storing the password of the user.
Key size	Keys may have one of the following lengths: 128, 192, or 256 bits	The minimum size is 1024 bits; the key size must be a multiple of 256.	The minimum key size is 160 bits.

AES looks like a good option given the flexibility in key size, as well as the way it functions is more seems more straight forwards

Feasibility analysis:

For compression, GZIP seems very feasible as it's libraries are already built into python, meaning no additional packages need to be downloaded. The addition of built-in libraries also means that when we code the compression algorithms it is most likely to be very fast and efficient, which will reduce power consumption. GZIP is also capable of compression any filetype, and it compatible on linux based text editors.

For encryption, AES allows for the smallest key size, which would increase efficiency and reduce storage space required. The only drawback is that this will result in a slightly less secure key.

Possible bottlenecks/issues:

For compression, the algorithm that GZIP uses may not be able to meet the requirements given to us, since using a predefined library limits our flexibility with the algorithm. This lack of flexibility could also cause an issue if the algorithm cannot process large sample sizes of data efficiently enough. This will possibly cause too much power consumption and result in failed ATPs

For encryption, as mentioned above, using the smallest encryption key size for the AES algorithm may cause security risks since it will be the easiest to decipher. Also, if the buoy is constantly sending and receiving data, this constant encryption and decryption can result in the system losing track of the keys and this may result in substantial errors in transmission and security.

Subsystem Design:

Our current subsystems are:

- Data capture as a .text file from the IMU

- Data compression algorithm in python
- Data encryption on python

Current sub-subsystems:

- Gzip library in python for compression of data
- AES encryption algorithm
- File handling system the data that is converted to a .txt file.

Requirements for subsystems:

- IMU data needs to be writable to a text file
- Compression algorithm needs to allow for enough fourier coefficients to extracted, as well as to allow for the smallest possible packages to be sent.
- Encryption method needs to be efficient and fully functional. Encrypted data needs to be intact and readable after being sent off.
- The encryption cannot take longer than 10ms.
- The encryption algorithm must be able to run on Raspberry Pi using the ARM processor.

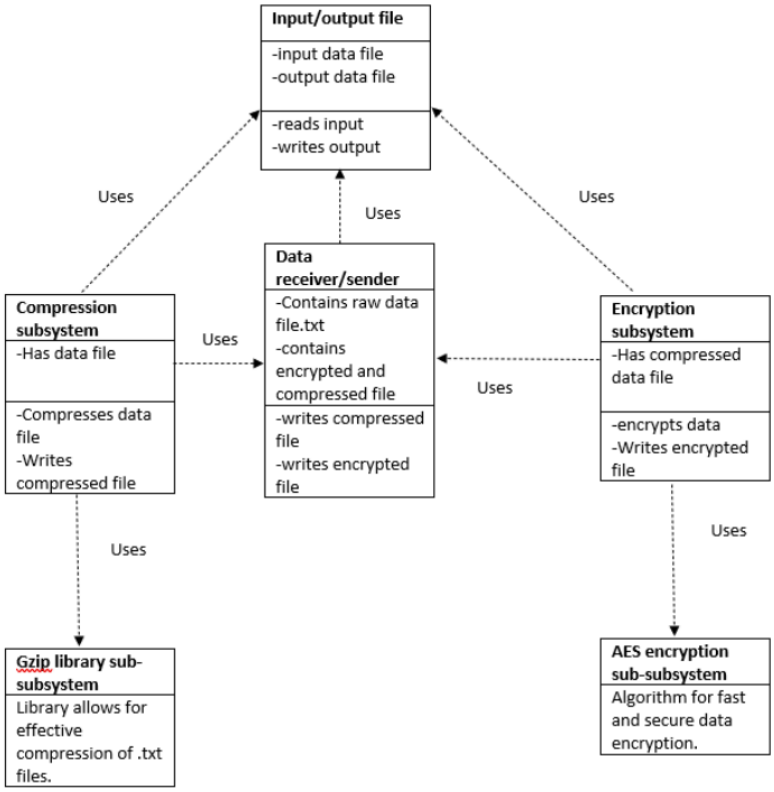
Subsystem specifications:

- Data file will be compressed using the Gzip library in python. At the very least, the file size needs to be a quarter of the original size.
- Data will be encrypted using the AES encryption algorithm. This will also be done in python. Encryption needs to be secure and lossless.
- Encryption key size is 128 bits

Subsystem inter-reactions:

The Raspberry Pi collects data from the IMU using the I2C communication protocol. This data will then be stored in a text file. The Data Compression Subsystem will be used to compress the data contained in the text file using one of the compression algorithms found in the Python libraries, and the output file will be sent to the Encryption Subsystem for encryption. The File will then be Encrypted and divided into manageable chunks that can be sent via Iridium Satellite.

UML diagram of the subsystems, and sub-subsystems:



ATPs:

AT001	Encryption Test
Evaluation type	Software Unit Test
Target	AES Encryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The encryption algorithm must be used to generate a key and use it to encrypt the data.
Pass condition	• The 128-bit key is generated and stored. • Data is encrypted. • Encrypted file is generated
Fail condition	• A key is not generated • Output file data looks the same as input data; meaning that the data is not encrypted. • File is corrupted • There is no output file. • Key size is incorrect.

AT002	Encryption Test
Evaluation type	Software Unit Test
Target	AES Decryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The decryption algorithm must be used to decrypt the data.
Pass condition	• The generated Key is stored safely and is used to decrypt data. • The output file is the same as the original file after compression.
Fail condition	• Output file data looks different to the original 'unencrypted' data • File is corrupted • There is no output file • The key fails to decrypt the file.

AT003	Compression Test
Evaluation type	Software Unit Test
Target	Compression Algorithm Subsystem
Test Protocol	Gzip Compression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new compressed file being generated.
Pass condition	• The text-file is compressed, i.e., the file size has been reduced significantly (45-55%)
Fail condition	• There is no output file. • The file output is not compressed, i.e., the file size is the same as the original file.

AT004	Decompression Test
Evaluation type	Software Unit Test
Target	Decompression Algorithm Subsystem
Test Protocol	Gzip Decompression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new decompressed file being generated.
Pass condition	• The text-file is decompressed, i.e., the file size has been restored to the original.
Fail condition	• There is no output file. • The file output is different to the initial file.

Development timeline:

✓ Mondays/Tuesdays/Fridays Had multiple meetings per week to discuss objectives and how we were going to split the project between us

✓ 21/08/2021: Decided on project management tool (Asana)

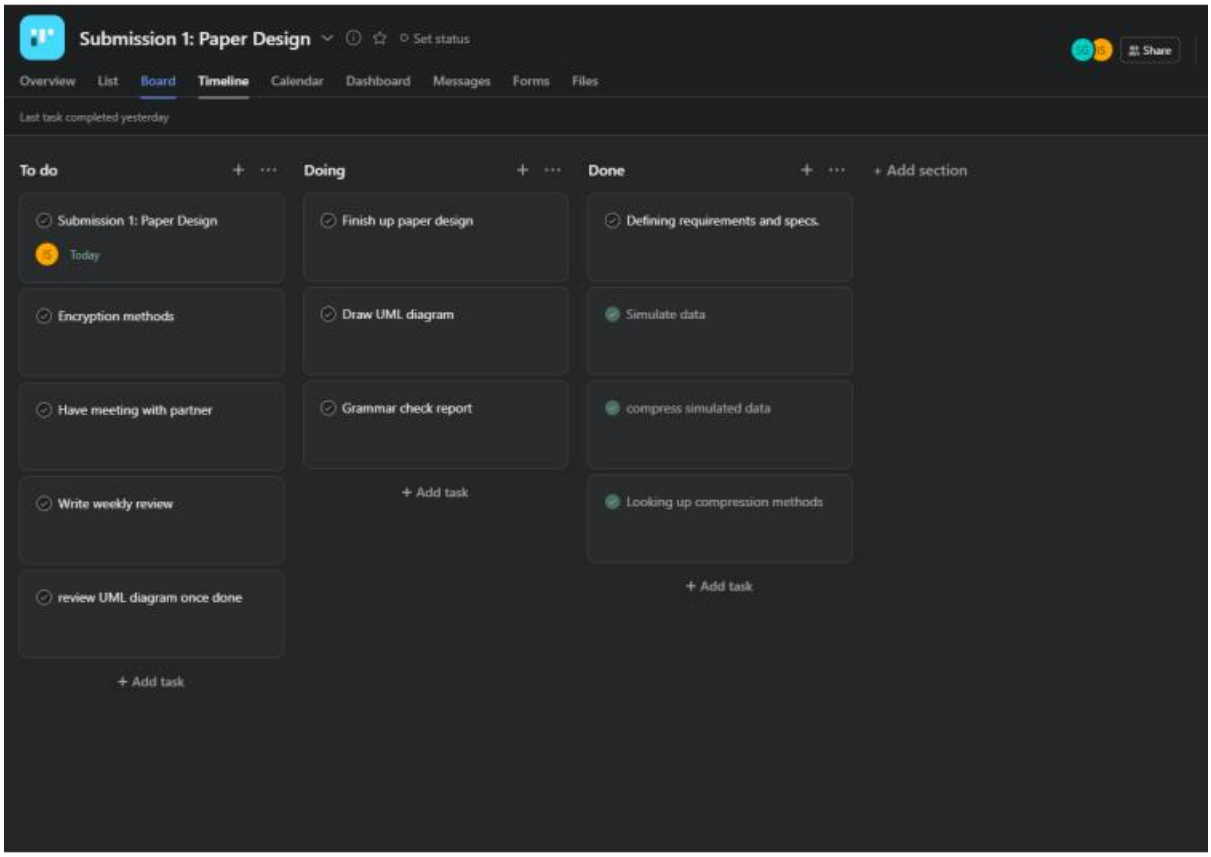
✓ 27/08/2021: Reviewed possible encryption and compression algorithms. Delegated each subsection to one another

✓ 03/09/2021: simulated IMU data in Matlab and successfully sent it to a .txt file and compressed it using Gzip in python.

✓ 04/09/2021: Met on MS Teams to start writing up the paper design for submission.

✓ 05/09/2021: Finished up paper design and prepared it for submission

Below is our Asana board:



Validation using Simulated Data:

Simulation based validation is important as it gives you a good idea of how the system is likely to perform in real world situations. It is a good way to gauge how well your system is currently likely to perform. It is also useful as it allows for you to control the type of data each system is tested on.

For the simulated data, there were two separate sets of data that we were able to test on. This data gathered from MATLAB, which simulated an IMU for us. The other set of data was a given on VULA in the form of a CSV file with a large amount of data entries. This CSV file was much larger than the data gathered from the MATLAB simulation. This was helpful as it allowed us to see the impact that a larger file would have on our algorithms, compared to the relatively small MATLAB data.

A snippet of the MATLAB data code is shown below (the diary command allowed for the output to be stored in a text file):

```
params = gyroparams

% Generate N samples at a sampling rate of Fs with a sinusoidal frequency
% of Fc.
N = 1000;
Fs = 100;
Fc = 0.25;

t = (0:(1/Fs):((N-1)/Fs)).';
acc = zeros(N, 3);
angvel = zeros(N, 3);
angvel(:,1) = sin(2*pi*Fc*t);

imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
[~, gyroData] = imu(acc, angvel);

figure
plot(t, angvel(:,1), '--', t, gyroData(:,1))
xlabel('Time (s)')
ylabel('Angular Velocity (rad/s)')
title('Ideal Gyroscope Data')
legend('x (ground truth)', 'x (gyroscope)')
get(0,'Diary')
```

This code produces a variety of parameters. These included:

- Resolution
- Noise density
- Temperature bias
- Acceleration Bias
- A graph showing Angular Velocity vs Time

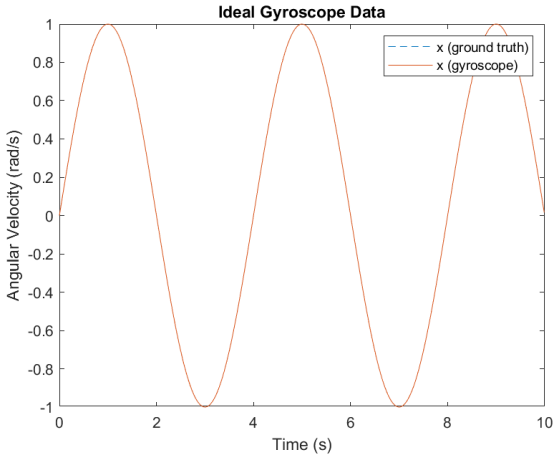
An image of the output plot and the output code gathered from the MATLAB simulation is shown below.

```
gyroparams with properties:

  MeasurementRange: Inf        rad/s
        Resolution: 0          (rad/s)/LSB
      ConstantBias: [0 0 0]    rad/s
  AxesMisalignment: [0 0 0]    %

     NoiseDensity: [0 0 0]     (rad/s)/√Hz
  BiasInstability: [0 0 0]     rad/s
       RandomWalk: [0 0 0]     (rad/s)*√Hz

         TemperatureBias: [0 0 0]     (rad/s)/°C
  TemperatureScaleFactor: [0 0 0]     %/°C
        AccelerationBias: [0 0 0]     (rad/s)/(m/s²)
```



The only major issue with this simulation, is that it only produces one set of data that never changes. The simulation gives similar data to the IMU that we will be using, but it is much smaller. It is for this reason that we believe the CSV file would produces a more realistic set of results.

Acquiring and testing the CSV files was simpler and it we just needed to download the files from file and run the algorithms on those files. A mentioned above, these files just contained a very large number of values for a wide set of parameters.

Experimental setup.

To test the two data test, we developed two separate programs. One program would compress and encrypt the data when run, and the second program would decompress and decrypt the program when run. We can call the program that compresses and encrypts program A, and the second program will be called program B. When program A is run, it reads the data file and then uses the GZIP library to compress all the data. The size of the new compressed file is checked against the ATPs. We also made sure that the contents of the compressed file were the same. After the data file is compressed, program A will then encrypt the newly compressed data file using AES encryption. In order to use this encryption method, it was necessary to first install the PyCryptoDome package for python. The program is able to function regardless of the type of file it is told to read, so it did not matter whether we gave it the .txt file or the .csv file. Once program A had been run, the datafile was checked to see if the data had indeed been properly encrypted.

Next, program B was run. Program B reads the compressed and encrypted data file and first decompresses. The size of the decompressed file was always checked. Program B then decrypted the decompressed data file using the AES key generated by program A. This decompressed and decrypted file was always checked to make sure it was the original size and contained all the original contents.

Each program implements time stamps in order to view the time taken for each stage. This made it easier to view which sub-systems were being the most effected by changes in the data size.

Results:

The results for various parts of the program's execution are tabulated below. Results are shown when the MATLAB data was used, as well as when the CSV IMU data was used. Each result shown below is the average taken from 10 samples. Overall, system performance was satisfactory will ATPs being met.

The following table shows the total completion time of each program, as well as the overall time to execute for the entire system. Individual subsystem results are shown later.

Overall system results for execution speeds.

	Compression + Decompression Time (ms)	Encryption + Decryption Time (ms)	Total time (ms)
MATLAB Data	28.72	10.9	39.62
CSV IMU Data	362.7	86.2	448.9

Compression and Decompression Results

	Compression Time (ms)	Decompression Time (ms)	Total time (ms)
MATLAB Data	2.9	8	10.9
CSV IMU Data	362.7	86.2	448.9

	Original File size	Compression size	Compression Rate
MATLAB Data	626 bytes	351 bytes	43.9%
CSV IMU Data	8650 KB	3686 KB	57.387%

These results are interesting, as the compression rate is entirely satisfactory for CSV IMU Data but falls a little bit short (threshold being 45%) with the MATLAB Data. Either the GZIP is less effective for smaller file sizes or the format of the data in the CSV file was easier to compress. Going forward we hope that the data simulated from the sense hat will compress in a similar way to the CSV data.

Encryption and Decryption Results

	Encryption Time (ms)	Decryption Time (ms)	Total time (ms)
MATLAB Data	19.7	9.02	28.72
CSV IMU Data	232.5	130.2	362.7

Each time, the data file was checked to see whether the encryption/decryption had been successful or not.

As shown, the file size has quite a large impact on the rate at which the algorithm was able to encrypt and decrypt the data. The impact is even larger than the impact on compression and decompression. The file size may even play a role in the compression rate of the GZIP algorithm. We still feel confident that the algorithm will be able to produce suitable encryption and decryption rates for larger files, although it is certain that larger files will take multiple seconds to be encrypted and decrypted. The key was also successfully created and stored in a file. When using a .txt file, the key was easily readable each time. They encrypted data that was written to a text file to be read, consistently showed the encrypted data when encrypted, and then showed the original data whence decrypted. The only concerning result was the compression rate for the small MATLAB data file.

Validation using a Different IMU:

Need for Hardware based validation:

Hardware based validation is important as it allows you to know how well the hardware part of your system is functioning. With so many important parameters being recorded, it is important to know how well the system is responding to physical changes. It is also important to be able to see how

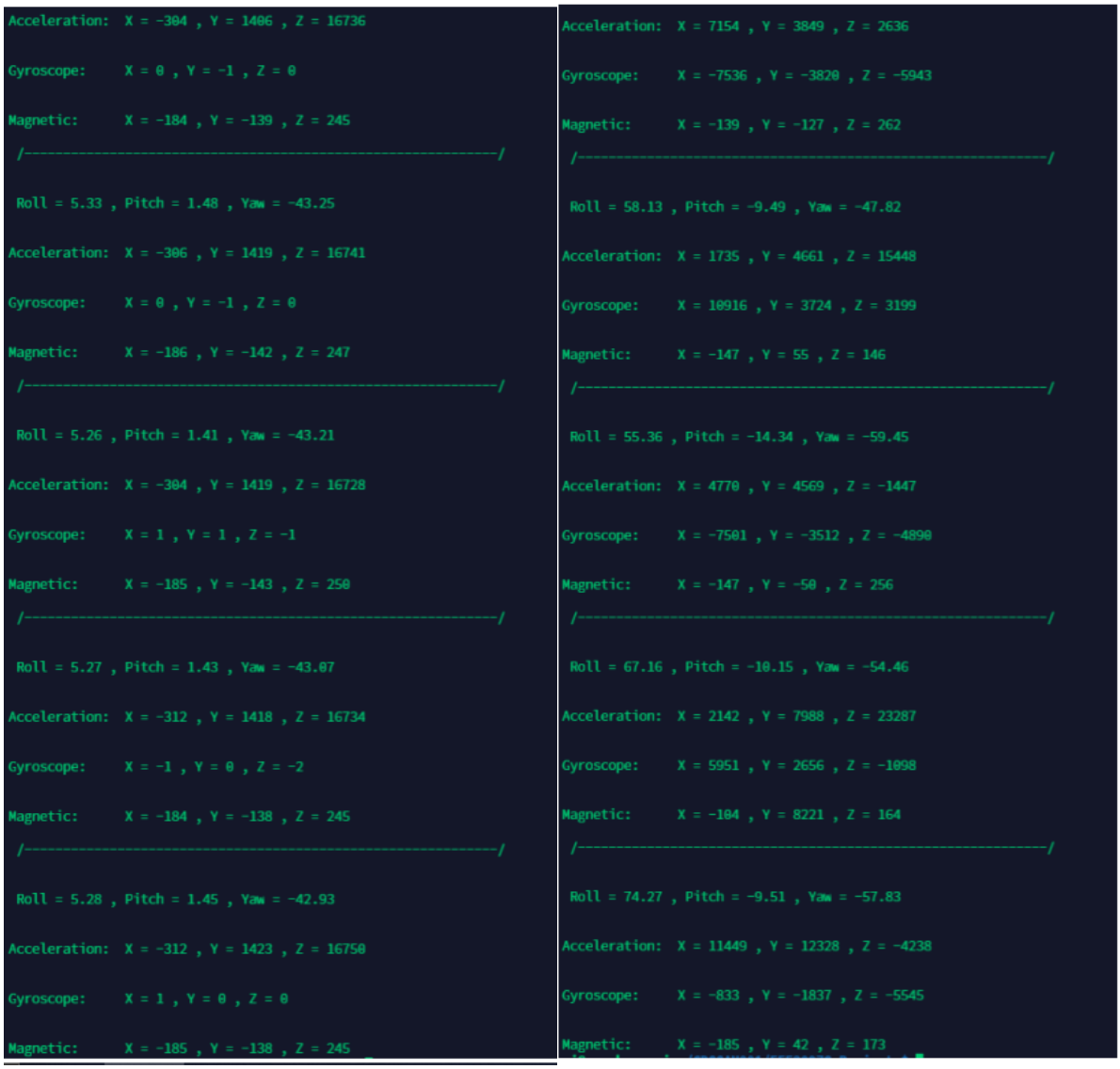
well the code is interacting with the physical components. For example, you need to be able to see if the IMU's pitch, roll, and yaw respond in the manner in which they are suppose to when the IMU is moved in the necessary way.

Firstly, it is important to note that the ICM used on the sense Hat is slightly different to the ICM on the shark buoy. The sense Hat uses the ICM-20948 and the shark buoy uses the ICM-20649. Below I have listed some of the key differences between these two.

ICM20948 (sense Hat)	ICM20649
3-axis gyroscope goes up to +2000dps	3-axis gyroscope goes up to +4000dps
3-axis accelerometer goes up to +16g	3-axis accelerometer goes up to +32g
3-axis magnetometer	-
Barometer, Temperature, Humidity	-
Resolution: 12-bits	On-Chip 16 bit ADCs
I2C	7MHz SPI or 400KHz I2C interface

Almost all of the hardware capabilities are the same between the two IMUs. The IMU in the shark buoy is likely to be slightly more accurate. This means that when adapting the sense Hat IMU to the bouy, we can simply make sure to use less samples, and to put less physical stress on the IMU. Considering the stress testing that we plan to put on the IMU through, this the accuracy between these two ICM will be negligible.

As mentioned in more detail below, the IMU produces data using python code designed for the sense Hat, and using the ICM-20948. When the program is running, it repeatedly reads a bunch of different data and prints it to the screen. The type of information that it gives is further described below. The format in which the IMU prints the data is shown below with two images. The image on the left shows some IMU output when the IMU is not being moved, while the image on the right shows the IMU output when the IMU is being moved around a lot.



The functionality of the IMU was also testing in two other ways. It was testing when being put Tupperware which was places in a body of water, as well as it was tested when a loud noise when played next to it. All tests yield the expected results.

There were a few steps involved with acquiring and using the IMU data for testing.

- First, we needed to plug the sene Hat onto the Pi and then download available code that would be able to simulate an ICM-20948.
- Running the ICM program gave out a constant stream of data with a multitude of parameters. These included:
    - Roll, Pitch, and Yaw.
    - Acceleration coordinates
    - Gyroscope coordinates
    - Magnetic coordinates
- The number of samples taken was varied each time, so that we could test on smaller and larger data sets to see how the algorithms dealt with the change in data sizes.
- Once the program was running, the first task was to send the data to a text file.

- Once the data was in a text file, it was easy to test the encryption and compression algorithms on the text file. Results on this testing are explained later in the report.

Experimental Setup:

The overall system:

Firstly, since the python code had only been used on a windows PC thus far, it needed to be transferred over to the Pi and edited to accommodate this. It was also necessary to download PyCryptoDome packages through the terminal. Once this had been done, all we needed was the data in a nano file. This was done by taking 4 different sets of data from the IMU (done using a for loop) and sending it do a nano file called 'myoutput'. These 4 sets contained 100, 200, 300,500 samples respectively. All 4 data sets were tested on for comparisons against each other. These comparisons are purely to see and test execution times for our algorithms on larger files compared to smaller sizes. Most importantly, we want to be able to see whether larger file sizes will affect the compression rate of the files. All results seemed to differ quite a bit when using a larger file size with the previous IMU systems, so we hypothesise that the same will be true when testing on our hardware.

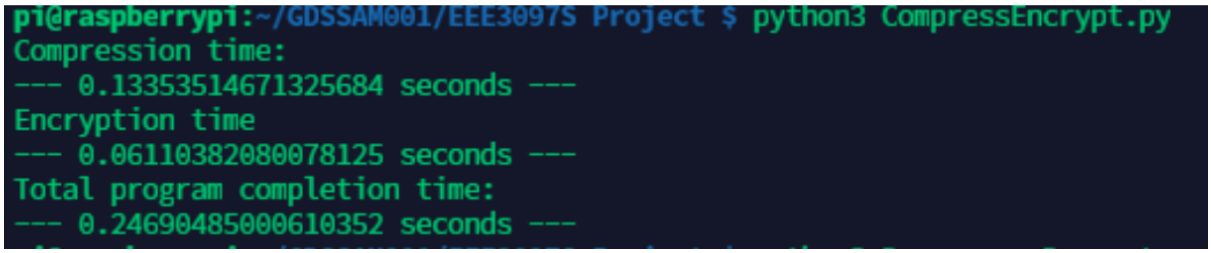
The IMU code used to generate the samples used an ICM20948. This code is specifically designed to be able to read data from the sense Hat. The compression and encryption sections were run as one file and the decompression and decryption sections were run as another file. Running both programs on the data file showed success in all subsystems. There was also a third program used call CEDD. This program contained functions for each subsystem and allowed for them to called individually using a prompt. This was useful for quick testing of each subsystem. However, for the most part, we used the two separate programs. File size was always checked for both the compressed and decompressed files.

As covered above, we also needed to make sure that the IMU and the IMC code was functioning properly. Running the ICM code continuously printed data reading to the screen. Tilting the IMU in the ways that measure roll, pitch, and yaw. We were able to gather data information for each of the parameters by doing this.

Total execution time was easy to acquire, as we just added up the total execution time of each program for each of the data sets. This is better shown below.

The compression and encryption program:

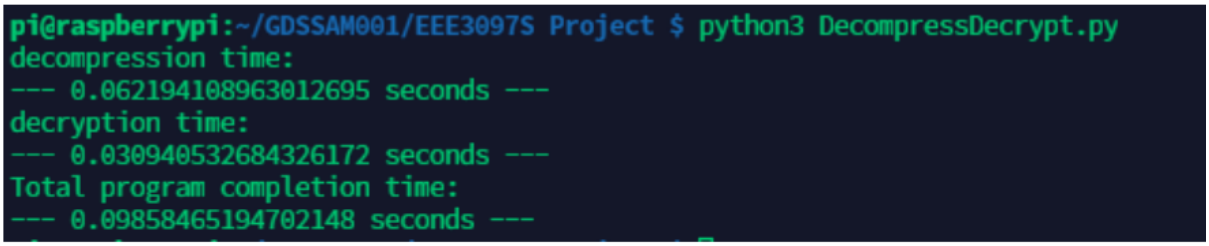
Compression and encryption are being run as one individual program. Like before, the program made use of AES encryption which is included in the PyCryptoDome package. The key is also stored in a nano file in the same directory. Again, the program makes use of Gzip to compress the data files. The program has time stamps after each subsection to determine the time taken for each sub system. This includes the time taken for compression, encryption, and the total program completion time. An image of how this looked is shown below. The example of the image below was taken when using 100 samples.



The data file was checked after each execution to make sure that the program was functioning as intended. This block is intended to take in IMU data in the form of a nano file and give out the data in the form of a compressed nano.gz file. Each time the program was run, I made sure to check the compressed file to see that the contents were the same. I also made sure to check that the file had been properly encrypted after the encryption was run. It was also necessary to make sure that the AES key was written properly.

The decompression and decryption program:

This program was executed in the same manner as the first. The decompression is once again using Gzip to function. The decryption locates the key in the local nano file and uses that to pad the file and decrypt it. Below is an image of taken after running the decompression and decryption program.



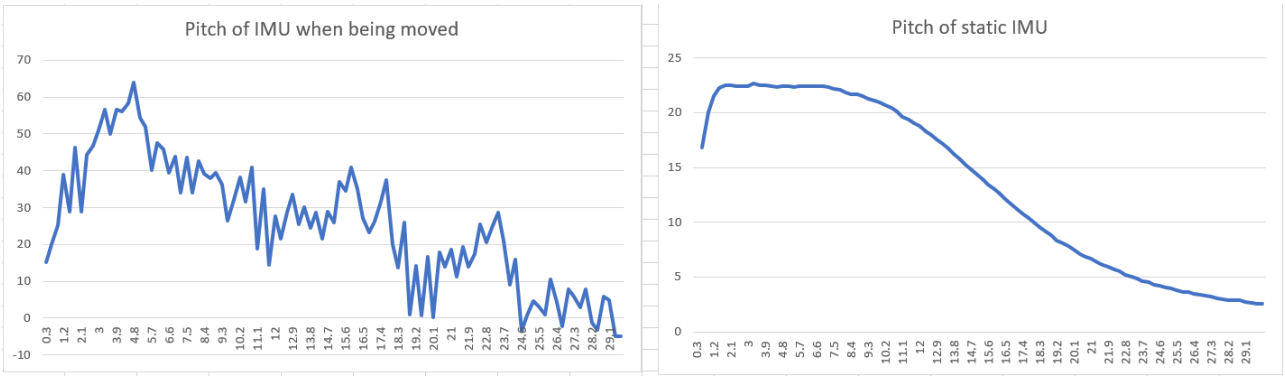
Running the program after running the compression and encryption program always successfully yielded the original file and its results. This block is supposed to take in the compressed and encrypted data in the form of a nano.gz file and return the original data in a nano file. The size of each file is checked before and after compression after each execution. The contents are also checked to make sure they are the same as the original. SCXITU002 is responsible for this block.

Results:

Results were promising, especially in the case of compression. Compression rate had the largest change from our simulated IMU results. The compression increased by 30-40%. Further details of this are shown below. Overall execution times did show to be slightly longer than before, but this makes sense since we are both testing a larger data set than before, as well as the Pi only having a single core to run on.

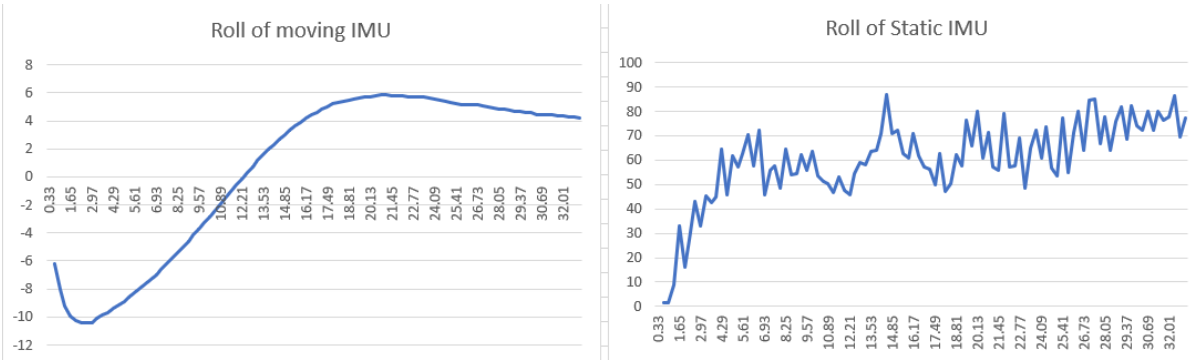
Firstly, are the results from testing the functionality of the sense Hat. Below are graphs plotted from data relating to the pitch, roll, and yaw of the IMU both when it was still and when it was being moved around to test for the specific parameter. The results are taken with each reading happening roughly every 0.333 seconds.

First is the plot of the pitch when it is being moved and not moved. Pitch was tested by pointing the IMU up and down.

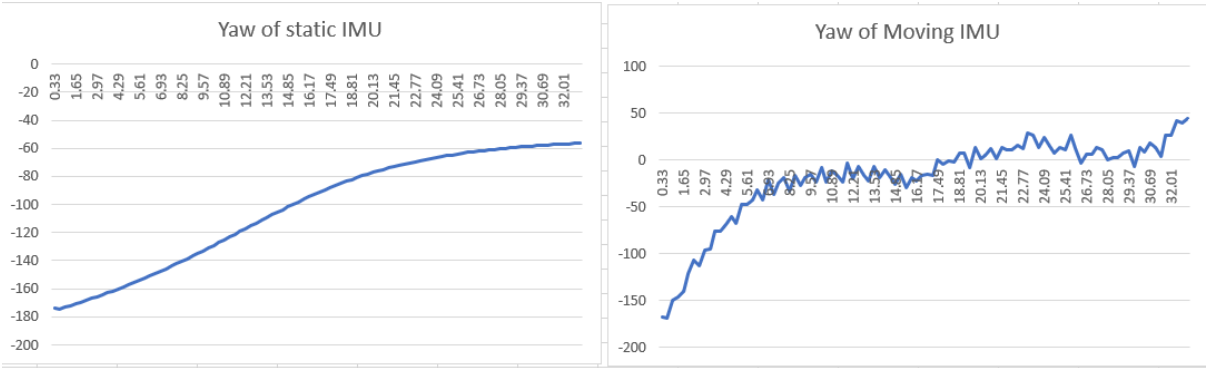


This clearly shows that the IMU responds to changes in the pitch. In each case, the pitch approaches zero as the samples reach their specified amount (100 in this case). We are unsure why this is happening.

Next is the graph of the Roll. Roll was tested by moving the IMU on its sides.



Next is the response of the change in Yaw. Yaw was tested by moving the IMU horizontally.



Encryption and compression results:

As mentioned above, data was taken from a nano file of 100, 200, 300, and 500 samples. Our ATPs have been slightly adjusted to accommodate for the slower performance. Each time the encryption key is successfully created and stored safely, as well as being successfully used to decrypt the data again, which passes our ATPs for encryption.

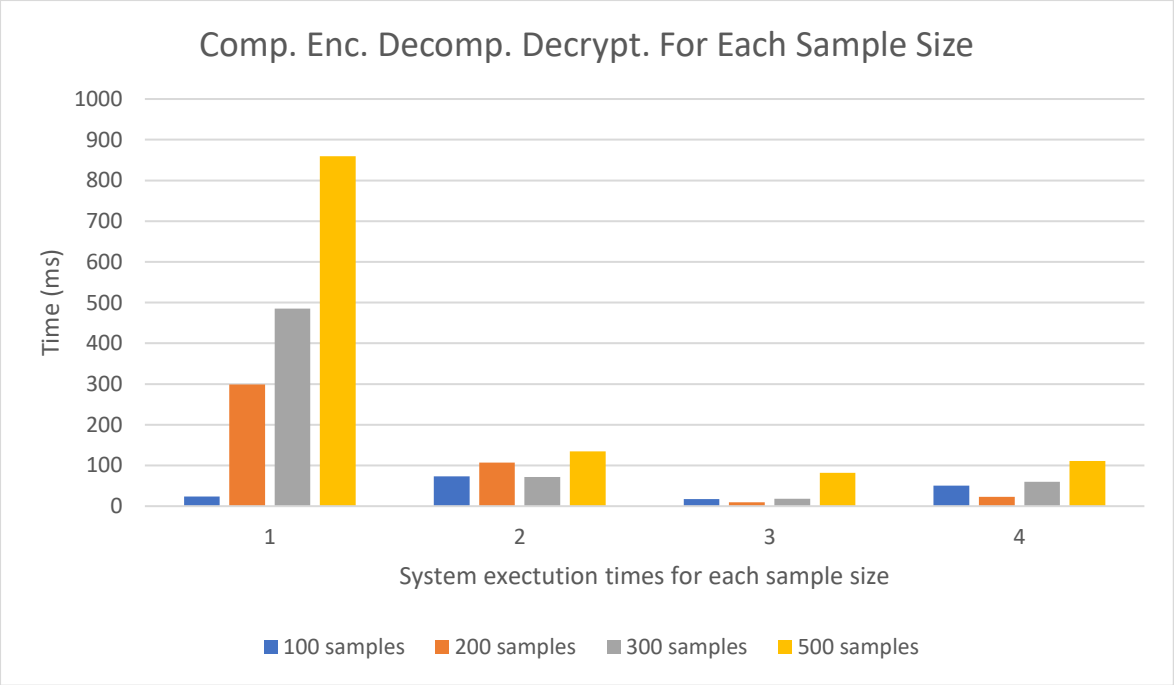
	Compression	Encryption	Total Time
100 samples	23.87ms	73.01ms	96.88ms
200 samples	298.93ms	106.68ms	405.61ms
300 samples	485.52ms	71.75ms	554.27ms
500 samples	859.47	134.83ms	994.3ms

Decompression and decryption results:

	Decompression	Decryption	Total Time
100 samples	17.38ms	50.56ms	67.94ms
200 samples	9.80ms	23.3ms	33.10ms
300 samples	18.54ms	59.58ms	78.12ms
500 samples	81.79ms	111.19ms	192.98ms

Below is a bar graph of each sub-system's performance as the sample size of the data file increases. Below shows how the numbering on the horizontal axis relates to the system being tested.

- 1 – Compression time
- 2 – Encryption time
- 3 – Decompression time
- 4 – Decryption time



Decompression and Decryption times are less important in these experiments, as in most real world scenarios, the data would be decompressed and decrypted on a more powerful device, such as a PC, instead of being locally on the Pi, like we have done here.

Below is a table showing the results from compressing the initial file sizes of each sample size.

	File size (bytes)	Compressed size (bytes)	Compression rate (%)
100 samples	25765	2861	88.9
200 samples	51317	4453	91.32
300 samples	76955	6090	92.01
500 samples	127920	9055	92

The results above are satisfactory and meet our requirements. Our compression far surpasses the required amount in both our requirements, and our ATPs, which only needed the compression to compress by 45-55%

Consolidation of ATPs and Future Plan:

Below are comprehensive tables of all out ATPs

Table 1: Acceptance Test Protocol for Encryption Algorithms

AT001	**Encryption Test**
Evaluation type	Software Unit Test
Target	AES Encryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The encryption algorithm must be used to generate a key and use it to encrypt the data.
Pass condition	• The 128-bit key is generated and stored. • Data is encrypted. • Encrypted file is generated
Fail condition	• A key is not generated • Output file data looks the same as input data; meaning that the data is not encrypted. • File is corrupted • There is no output file. • Key size is incorrect.

Table 2: Acceptance Test Protocol for Decryption Algorithm

AT002	**Encryption Test**
Evaluation type	Software Unit Test
Target	AES Decryption Algorithm Subsystem
Test Protocol	The AES Encryption Algorithm must be imported from python libraries. The decryption algorithm must be used to decrypt the data.
Pass condition	• The generated Key is stored safely and is used to decrypt data. • The output file is the same as the original file after compression.
Fail condition	• Output file data looks different to the original 'unencrypted' data • File is corrupted • There is no output file • The key fails to decrypt the file.

Table 3: Acceptance Test Protocol for Compression Algorithm

AT003	**Compression Test**
Evaluation type	Software Unit Test
Target	Compression Algorithm Subsystem
Test Protocol	Gzip Compression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new compressed file being generated.
Pass condition	• The text-file is compressed, i.e., the file size has been reduced significantly (45-55%)
Fail condition	• There is no output file. • The file output is not compressed, i.e., the file size is the same as the original file.

Table 4: Acceptance Test Protocol for Decompression Algorithm

AT004	**Decompression Test**
Evaluation type	Software Unit Test
Target	Decompression Algorithm Subsystem
Test Protocol	Gzip Decompression Algorithms should be imported from Python libraries. Gzip Compression algorithm must be used to compress the data from the IMU, resulting in a new decompressed file being generated.
Pass condition	• The text-file is decompressed, i.e., the file size has been restored to the original.
Fail condition	• There is no output file. • The file output is different to the initial file.

Table 5: Acceptance Test Protocol for Specification Test

AT005	**Specification Test**
Evaluation type	Parameters
Target	Hardware components
Test Protocol	Evaluate ICM-20948 specifications from the datasheet and check if these meet the ones requirements for the system that we are building.
Pass condition	• The specifications meet the desired requirements
Fail condition	• Specifications do not meet the desired requirements.

Table 6: Acceptance Test Protocol for data Capturing Test

AT006	**Data Capturing Test**
Evaluation type	Software Unit Test
Target	Data Capture Subsystem
Test Protocol	The data from the IMU should be recorded and saved as a nano file on the raspberry pi to allow for compression and encryption.
Pass condition	• Data is captured and saved in a text file.
Fail condition	• There is no output file.

Table 7: Acceptance Test Protocol for System Robustness

AT007	**System Robustness**
Evaluation type	Software
Target	System and Subsystems
Test Protocol	Connect the ICM-20948 to the Raspberry Pi and take measurements and store them in a nano file. Moreover, run the entire system (AES encryption and Gzip algorithms), i.e., producing the compressed and Encrypted files. Run this process 100, 200, 300, and 500 times.
Pass condition	• The Raspberry Pi can successfully complete the consecutive cycles.
Fail condition	• Failure to run the cycles. • Failure to encrypt the data • Failure to compress the data

Table 8: Acceptance Test Protocol for Power Consumption Testing

AT008	**Power Consumption Testing**
Evaluation type	Hardware
Target	Power System
Test Protocol	Use an inline power meter, and Retropie Emulator to get the power usage while the system is running.
Pass condition	• The device uses power of less than 60Wh
Fail condition	• Electrical Failures • Device uses more 60Wh when running.

AT009	**System Execution Time**
Evaluation type	Software
Target	System and Subsystems
Test Protocol	The simulation data should be used to run the encryption and compression algorithms. A timer may be used to check the execution time.
Pass condition	• The execution time is 1 second. • The Execution time may take longer for larger files; depends on the size of the data. (Gigabytes of data)
Fail condition	• Execution time takes longer than 1 second for the simulated data.

Table 9: Full System Acceptance Test Protocol

Full System Acceptance Test Protocols				
Test ID	Fully Satisfied	Partially Satisfied	Not Satisfied	Comment
AT001	✓			
AT002	✓			
AT003	✓			
AT004	✓			
AT005	✓			
AT006	✓			
AT007	✓			
AT008			✓	We have not implemented these systems
At009	✓			

For this to work for the sharc-bouy team, they would just need to implement a system to send the compressed and encrypted data to an external machine to be decrypted and decompressed. Also they might need to add in a system that can measure power usage.

Conclusion:

We are happy with the overall performance of our project. Despite initial problems with compression rates, we were able to get a compression algorithm that functioned above expectation. The encryption algorithm works flawlessly, with no data being lost in the process and the key is always readable and able to decrypt the file. We were successful in both simulating IMU data, as well as gathering IMU data from an actual IMU in the form of the sense Hat. We were fully able to utilize this data and determine executions times, and various graphs pertaining to the information returned by the IMU.

Our system is almost in a fully functional state, and we are satisfied all the systems, subsystems, and sub-subsystems. The only thing this would need to do now for it to work for the sharc-buoy would be for it to transmit the data off the pi and onto another device such as a PC.