EEE3097S
Final Report

Team Members:

Yondela Matwele (MTWYON001)

Teddy Muba (TDDMUB001)

1. Admin documents

Contribution Table

Contribution Discussion	Section Numbers	Page Numbers
Yondela		
Worked on the admin section of this document.	1	2-4
Within the requirement analysis worked on the overall system requirements and the compression/decompression section of the requirements, including the specification sections as well.	2.1-2.2, 2.4-2.5	5-6, 7-11
The methodology and comparison of algorithms when it comes to the compression subsystem.	3.2.1	14-15
This section covers the compression block of the system block diagram.	3.2.3.1	19-20
This section covers validating the system using simulated data, the setup of the experiments and the results of those experiments.	4.1-4.2, 4.3.1-4.3.2	22-23
This section covers validating using the actual IMU data that has been created by the IMU sensor.	5.1-5.2	30-36
This section covers the conclusion of the report from the compression side of things	7	40

Teddy		
Requirement Analysis: Worked on the encryption requirements	Section 2.3	7-8
Specifications: Worked on encryption specifications	Section 2.6	12-13
Paper Design**:** Methodology & Algorithms Comparison - Worked on the encryption Blocks, project system block diagram, UML Use Case Diagram	Section 3	17, 18, 19, 21
Experiment Setup: Experiments/Simulations of the compression and decompression blocks	Section 4	23, 24, 25, 26, 27, 28
Result: the encryption and decryption results	Section 5	36
Consolidation of ATPs and Future Plan: Encryption & Decryption	Section 6	38, 39
Conclusion	Section 7	40
References	Section 8	41

Table 1: Contribution table

Snapshot of project management tool

Link of project management: https://trello.com/b/qxBZTpx9/eee3097s

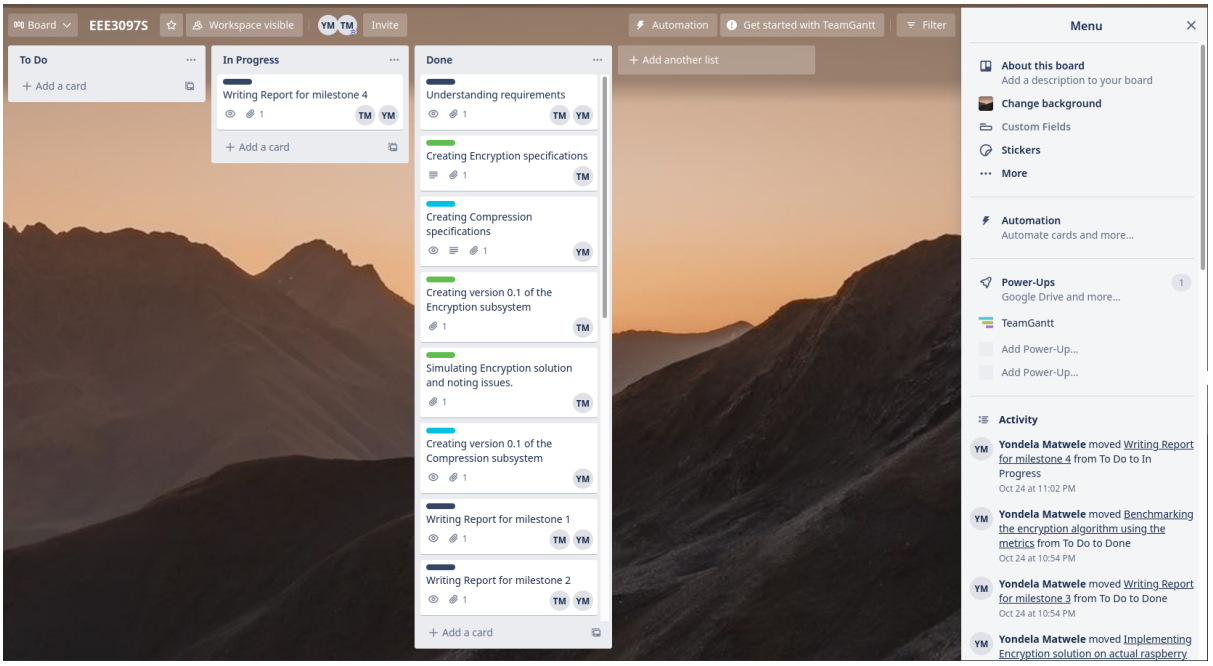


Figure 1: Snapshot of project management tool

Github Page link

https://github.com/Yondela/EEE3097S_project

Timeline

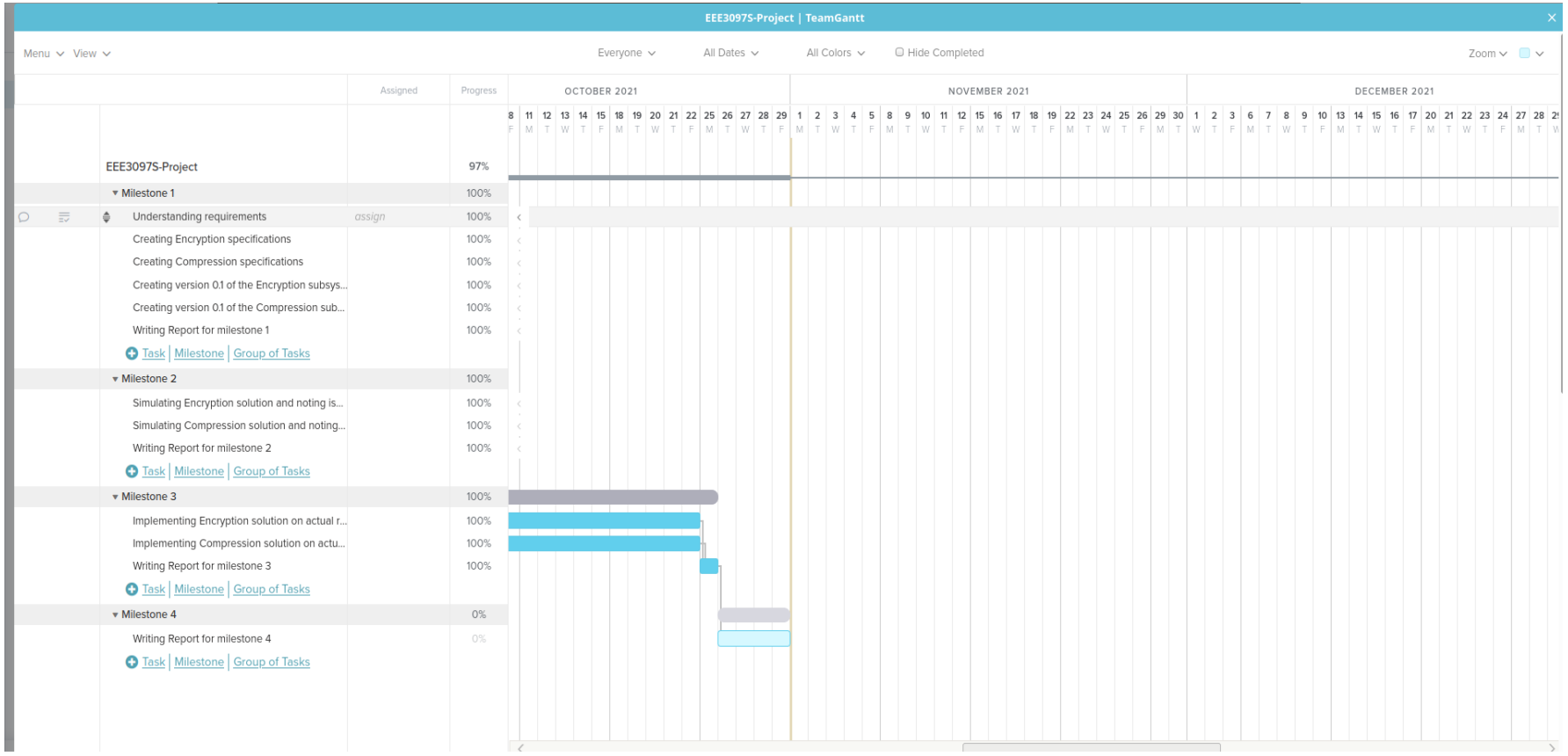


Figure 2: Timeline of the project

2. Requirement Analysis

Overall System Requirements

- Requirement 1.1: The IMU to be used is ICM-20649
    - This requirement means we are limited to the specific component that has been chosen to work for this project. According to the temperature ranges of the device it should be operational within the conditions that it will be operating in.
- Requirement 1.2: Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data.
    - This requirement focuses on the data oceanographers require. So it has to be at least 25% of the Fourier coefficients of the data; this is if the data does have some losses throughout the process.
- Requirement 1.3: In addition to reducing the amount of data we also want to reduce the amount of processing done in the processor (as it takes up power which is limited).
    - This requirement focuses on the power consumption which is critical due to the limited resources that are experienced in the environment that the buoy will operate in.
- Requirement 1.4: The data has to be compressed and encrypted when the IMU is active and then the data should be sent when the buffer is filled or just before the IMU goes into sleep mode.
    - This requirement focuses more on ensuring that power is used efficiently instead of having the entire system active all the time instead some parts will be in sleep mode when they are not needed so that the power consumption is not high to ensure that the battery usage is focused on what's needed at that time.
- Requirement 1.5: The sample rate should not be excessive.
    - This requirement focuses on ensuring that the power consumption is not too high due to the fact that when collecting data it doesn’t have to be such a high frequency that will consume too much power because of the high frequency instead if the sampling rate is lower then it is better.
- Requirement 1.6: The data doesn’t need to be sent in real time. There can be delays with it being sent.
    - This requirement focuses on defining the nature of the system where if it was real time then as the data is created there can be a small amount of delay before it gets sent however since the system is not real time some delays can be accepted meaning the data can be sent as batches instead of it being streamed.

Compression Requirements

- Requirement 1.7: The compression should not be power intensive.
    - This requirement focuses on the power usage of the compression algorithm. Since the system has limited power the algorithm needs to be power efficient where it doesn't use too much power.
- Requirement 1.8: The compression algorithm should be run autonomously.
    - This requirement focuses on the issues that can be caused if the algorithm needs manual intervention which is very rare and expensive to perform so if the algorithm was as autonomous as possible and it could self diagnose all the known issues and solve them then it would be better because it is not as accessible.
- Requirement 1.9: The compression data should be smaller than the original data.
    - This requirement focuses on the data that is going to be used which needs to be less than the original data because the cost of sending data packets is very high so then it would be cheaper to send smaller packets of data because it would be smaller.
- Requirement 1.10: The compressed data should be decompressed on the other side.
    - This requirement focuses on ensuring that the data can be decompressed on the other side once it is received due to the fact that if the data can't be decompressed then it will be useless to the oceanographers.

Encryption Requirements

- Requirement 1.11: Provision of a manageable encryption key size.
    - This requirement refers to the random string of bits to be generated to specifically scramble the compressed data. The encryption keys will be created with algorithms designed to ensure that each key is unique and unpredictable.

- Requirement 1.12: Provision of a manageable decryption key size.
    - This requirement refers to the random string of bits to be generated to specifically unscramble the encrypted data. The decryption keys will be created with algorithms designed to ensure that each key is unique and unpredictable.

- Requirement1.13: Provision of confidentiality and data integrity to users.
    - This requirement refers to data encryption as a safeguard for confidentiality. We will be looking at restricting data access to authorized users while enabling data integrity with inherent encryption mechanisms to protect data at whether it is at rest and/or in transit.

- Requirement 1.14: Assuring compatibility of the encryption/decryption algorithm with the ICM-20649 IMU (sensor).
    - This requirement focuses on the alignment between the RaspberryPI specifications, the ICM-20649 IMU data generation with the encryption/decryption algorithm.
- Requirement 1.15: Encryption algorithms performance comparison.
    - This requirement refers to the fact that besides the principal goal guiding the design of any encryption algorithm which consists of security against unauthorized attacks, attention has to be paid on important concerns such as the performance and the cost of the choice of the algorithm to be implemented.

Overall System Specifications

Type of Requirement	**Requirements**	**Specifications**	**ATPs**
Component Requirement	**R1.1:** The IMU to be used is ICM-20649	**S1.1:** The IMU has to operate between -40 to +85 degrees celsius.	The IMU will be operated in the following temperatures to see if it still produces the same results when the same movements are conducted on the IMU. The temperatures are -20, -10, 0, 10, 20, 30, 40, 50, and 60 degrees. The IMU will be operated for 1 hour per temperature value to see if it does produce the same output when the same movements are conducted.
Processing Requirement	**R1.2**: Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data.	**S1.2**:The IMU has to work with I2C.	The IMU will be tested by comparing the output of the IMU when the SPI and I2C of the same frequency is used to see if it does produce

			the same output when the same movements are conducted on the IMU.
Data Requirement	**R1.3**: In addition to reducing the amount of data we also want to reduce the amount of processing done in the processor (as it takes up power which is limited).	**S1.3**: The data produced from the entire system should not be corrupted.	The IMU data will be checked for various datasets. The datasets that will be used are as follows, sin, cos, tan, gaussian noise, the trig functions that were used without noise superimposed with noise, and the actual data from the IMU. All the datasets will be compared to see if the data in the input side of the system and the output side of the system are the same and not corrupted.
Processing Requirement	**R1.1:** The IMU to be used is ICM-20649	**S1.4:** The IMU should be supplied 1.8 - 3.6 volts	The IMU will be tested to see if it is producing the same output for the different volt values of 1.8, 2.0, 2.5, 3.0 and 3.6 volts. The same movement will be done with all the various voltages that are used to check if the same values are generated by the IMU.
Data Requirement	**R1.5:** The sample rate should not be excessive.	**S1.5:**The system should produce enough data to not go above the 512	The IMU will be tested by increasing the frequency at which the data is

		bytes FIFO buffer provided by the IMU	produced which will help determine if the IMU can handle the maximum frequency at which the data is being sent to the raspberry pi. The raspberry pi will also cause delays to test if the buffer can hold 512 bytes of data.
Processing Requirement	**R1.1:** The IMU to be used is ICM-20649	**S1.6:** The IMU should be able to withstand 10000g of shock	The IMU will be tested by putting it in an environment where it will experience 10000g and it will be kept in the environment for about 15 minutes and then the system will run through a sequence of motion which will test if the IMU is still operational.
Processing Requirement	**R1.3:** In addition to reducing the amount of data we also want to reduce the amount of processing done in the processor (as it takes up power which is limited).:	**S1.7:** The overall system should not operate at 100% CPU utilization	This will be tested by having the system processing various streams of data continuously for 1 hour to ensure that it doesn't operate at 100% CPU utilization.

Table 2: Overall System Specifications

Compression specifications

Type of Requirement	Requirements	Specifications	ATPs
Processing Requirement	**R1.8**: The compression algorithm should be run autonomously.	**S1.7**: The compression algorithm should not require manual intervention for it to run.	This will be tested by having the algorithm dealing with various data sets and looking at if the data that is compressed is not corrupted and is compressed properly. This test will be run over 10 various data sets which will contain sin, cos, tan, gaussian noise, the trig functions with gaussian noise, the data set we were given with/out noise, and finally the data generated by our IMU with/out noise.
Data Requirement	**R1.9**: The compression data should be smaller than the original data.	**S1.8**: The compressed data has to be less than the original data when it comes to size.	This test will be run over 10 various data sets which will contain sin, cos, tan, gaussian noise, the trig functions with gaussian noise, the data set we were given with/out noise, and finally the data generated by our IMU with/out noise.

			The data will be compared for each one to ensure that the compressed data is less than the original data.
Data Requirement	**R1.10**: The compressed data should be decompressed on the other side.	**S1.9**: The data on the other side should be good enough in order to be decompressed on the other end and it should not be corrupted as well.	This test will be run over 10 various data sets which will contain sin, cos, tan, gaussian noise, the trig functions with gaussian noise, the data set we were given with/out noise, and finally the data generated by our IMU with/out noise. The data will be compared to check if the data has been corrupted.

Encryption specifications

Type of Requirement	**Requirements**	**Specifications**	**ATPs**
Data & Processing Requirement	**R1.12**: Provision of a manageable encryption key size.	**S1.12:**AES should have a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits	This will be tested using the numbers of cycles of repetition as follows: ● 10 cycles of repetition for 128-bit keys. ● 12 cycles of repetition for 192-bit keys. ● 14 cycles of repetition for

			256-bit keys.
Data & Processing Requirement	**R1.13**: Provision of a manageable decryption key. size. The data receiver should be able to view the original data by using decryption technique.	**S1.13:** AES should have a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits	This will be tested using a set of reverse rounds applied to transform ciphertext back into the original plaintext.
Processing Requirement	**R1.14**: Provision of confidentiality and data integrity to users.	**S1.14:** ● To encrypt, an implemented algorithm has to run and encrypt the plaintext using a generated subkey. ● Running the algorithm multiple times should generate different encrypted messages. ● To decrypt, an algorithm has to run and decrypt the decrypted text using the same subkey generated above.	This will be tested when going through the following steps: ● Powering ON the RaspberryPI and LCD display. ● Starting the OS by using start_x command. ● Opening terminal in the RaspberryPI desktop. ● Running the file in the terminal of the RaspberryPI desktop. ● Running the code and the algorithm will run and use the generated subkey to encrypt the file. ● The algorithm will continue running and use the generated subkey to decrypt the file.
Processing Requirement	**R1.15**: Assuring compatibility with the ICM-20649 IMU (sensor).	**S1.15:** The RaspberryPI specifications should be aligned to this requirement	This will be tested by: ● Initially enabling the ARM board

			by giving the power supply. ● Connecting keyboard, mouse, lcd display to the ARM. ● Opening Qt/or any other software in the LCD display. ● Building, compiling, running the code in Qt/or any other software ● Selecting cover file, secret file from the SDcard

3. Paper Design

Concept

In this project we are thinking of designing an ARM based digital IP using a Raspberry-Pi to compress and encrypt an IMU environmental data. This IMU is one of the crucial sensors living in a generic flexible buoy currently being installed on the pancake ice in the Southern Ocean. The IMU will be providing information about the ice as well as the wave dynamics. Considering the cost that comes with data transmission using Iridium, the developed idea will revolve around buffering and sending as much of the IMU data as possible. This will be implemented by firstly compressing and then encrypting the IMU data.

The required process will consist of designing an IP for an virtual ICM-20649 IMU (sensor). The compression process should lead to a designed digital IP that should allow oceanographers to be able to extract at least 25% of the Fourier coefficients of the data. This

will be implemented while considering the reduction of the amount of processing done in the processor for power consumption limitations.

In the following lines, we will be digging into all possible requirements analysis (paper design) to be converted into hard specifications for the ARM based digital IP using a Raspberry-Pi. The IP will be divided into sub-systems to specify the inter and intra sub-system interactions. Finally, acceptance criterias and test cases will be put in place to demonstrate if the design is in phase with required specifications.

Methodology & Algorithms Comparison

3.1.1. Compression Block

The compression algorithm that is best for the specified project would be the gzip compression algorithm. The reason the gzip algorithm is the better option when compared the bzip and xz compression algorithms is because it has the lowest compression time (runtime), this helps since we do not want to spend a lot of time using the CPU for compressing the data due to the power constraints that the buoy is experiencing. The gzip compression algorithm has a reasonable compression ratio where it is lower than half of the original data size for the amount of time it takes to compress the data. The decompression aspect of the gzip algorithm is also fairly low when it comes to power consumption due to the small amount of time it takes to decompress the data when compared to compression algorithms like the bzip and xz compression algorithms.

The gzip compression algorithm is also good since it is a lossless compression because we do not want to lose any sort of data recorded due to the fact that if there was some loss in during the compression it would really temper with the information since it was recorded in the time domain and the fourier coefficients are only processed after converting the time domain signal to frequency domain.

The reason the xz compression is not a great choice for this task is because it takes the longest time to compress the data which will be a problem due to the constraint of power. However it does offer a higher compression ratio. The reasons the bzip2 compression is not a great choice for this project is the compression time it takes is far too long when compared to gzip however bzip2 has a better compression ratio than gzip.

The table below indicates the size in bytes of the linux-3.18.19.tar file after compression, the first column numbered 1.9 shows the compression level passed in to the compression tool.

	gzip	**bzip2**	**xz**
1	153617925	115280806	105008672
2	146373307	107406491	100003484
3	141282888	103787547	97535320
4	130951761	101483135	92377556
5	125581626	100026953	85332024
6	123434238	98815384	83592736
7	122808861	97966560	82445064
8	122412099	97146072	81462692
9	122349984	96552670	80708748

The following table shows the amount of time taken to compress the above mentioned linux-3.18.19.tar

	gzip	**bzip2**	**xz**
1	13.213	78.831	48.473
2	14.003	77.557	65.203
3	16.341	78.279	97.223
4	17.801	79.202	196.146
5	22.722	80.394	310.761
6	30.884	81.516	383.128
7	37.549	82.199	416.965
8	48.584	81.576	451.527
9	54.307	82.812	500.859

The final compression algorithm used due to the need to code the algorithm from scratch was the lz77 algorithm which is part of the gzip algorithm.

3.1.2. Encryption Block

On the other hand, The methodology of the encryption algorithms will be based on cryptographic methods. Three alternative cryptographic methods were assessed in some past papers experiments and analysis.

These cryptographic methods are RSA (Rivest-Shamir-Adleman), AES (Advanced Encryption Standard) and DES (Data Encryption Standard) as tabulated below:

3.1.2.1. General Comparison between AES, DES and RSA

Factors	**AES**	**DES**	**RSA**
Development year	2000	1977	1978
Key Size	128, 192, 256 bits	56 bits	>1024 bits
Block Size	128 bits	64 bits	Minimum 512 bits
Ciphering & deciphering key	Same	Same	Same
Scalability	Not Scalable	It is scalable algorithm due to varying the key size and Block size	Not Scalable
Algorithm	Symmetric Algorithm	Symmetric Algorithm	Asymmetric Algorithm
Encryption	Faster	Moderate	Slower
Decryption	Faster	Moderate	Slower
Power Consumption	Low	Low	High
Security	Excellent Secured	Not Secure Enough	Least Secure
Deposit of keys	Needed	Needed	Needed
Inherent Vulnerabilities	Brute Force Attack	Brute Forced, Linear and differential cryptanalysis attack	Brute Forced and Oracle attack
Key Used	Same key used for Encrypt and Decrypt	Same key used for Encrypt and Decrypt	Different key used for Encrypt and Decrypt
Rounds	10/12/14	16	1
Stimulation Speed	Faster	Faster	Faster
Trojan Horse	Not proved	No	No

Hardware & Software Implementation	Faster	Better in hardware than in software	Not Efficient
Ciphering & Deciphering Algorithm	Different	Different	Same

Table 3: Comparison between AES, DES and RSA

- **Comparisons of DES, AES and RSA of Encryption and Decryption Time**

S.NO	Algorithm	Packet Size (KB)	Encryption Time (Sec)	Decryption Time (Sec)
1	AES	153	1.60	1.00
	DES		3.00	1.10
	RSA		7.30	4.90
2	AES	196	1.70	1.40
	DES		2.00	1.24
	RSA		8.50	5.90
3	AES	312	1.80	1.60
	DES		3.00	1.30
	RSA		7.80	5.10
4	AES	868	2.00	1.80
	DES		4.00	1.20
	RSA		8.20	5.10

Table 4 : Comparisons of DES, AES and RSA of Encryption and Decryption Time

By analyzing table-4, Time taken by RSA algorithm for both encryption and decryption process is much higher compared to the time taken by AES and DES algorithm.

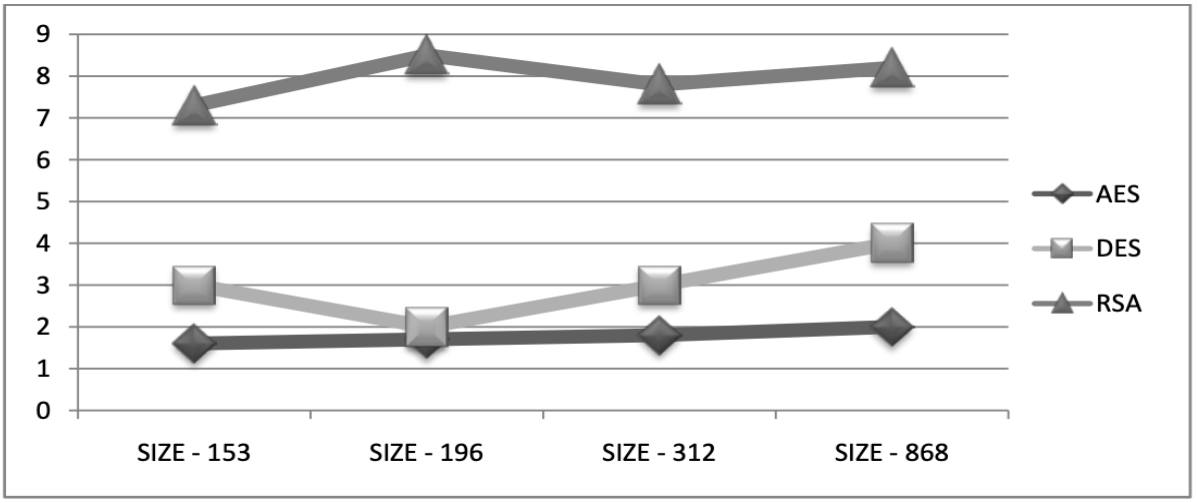


Figure 3 : Comparison of Encryption Time among AES, DES and RSA

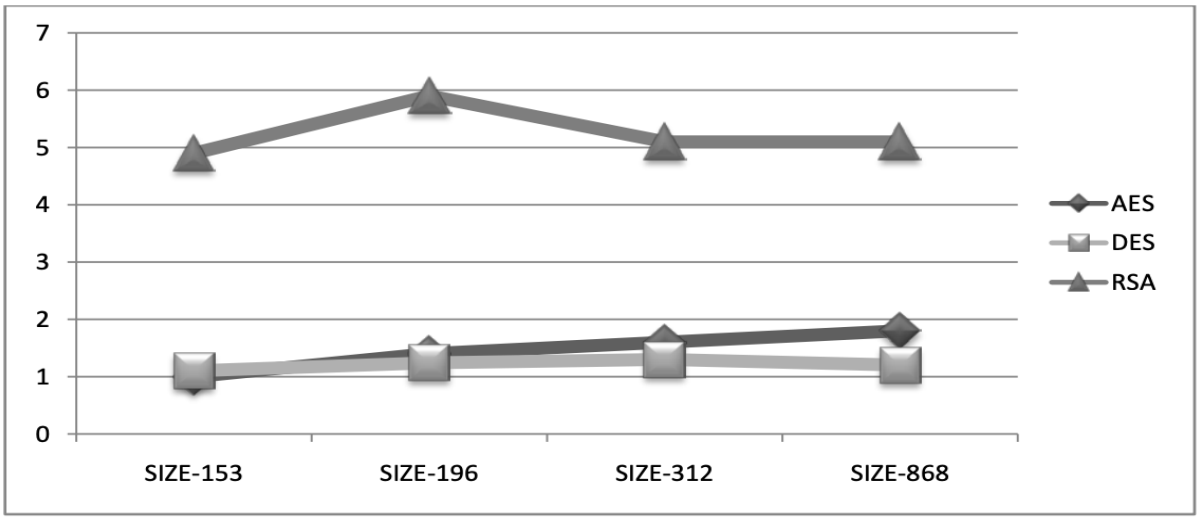


Figure 4 : Comparison of Decryption Time among AES, DES and RSA

By analyzing Fig-3 , Fig-4 which shows time taken for encryption and decryption on various sizes of file by three algorithms. The RSA algorithm takes much longer compared to the time taken by the AES and DES algorithm. AES and DES algorithms show very minor differences in time taken for the encryption and decryption process.

Our future work will focus on developing and analysing the AES cryptographic algorithm. It will include experiments on the ICM-20649 IMU (sensor) data to be compressed and encrypted.

3.1.3. Project System Block Diagram

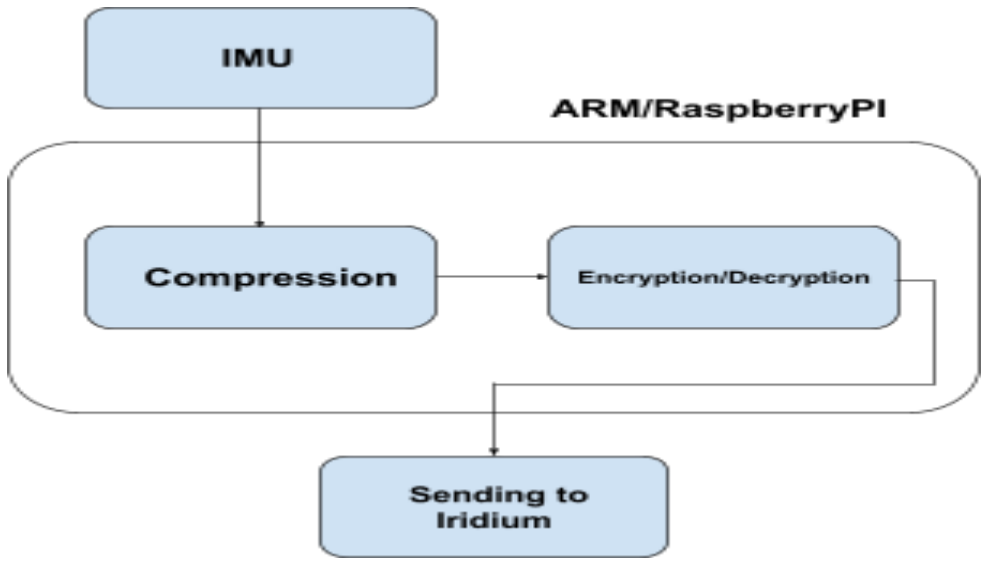


Figure 4 : System Block Diagram

3.1.3.1. Compression Block

For the compression and decompression subsystem the design has been changed from the previous design that was stated in the paper design. In the paper design the algorithm that was going to be used was going to be the gzip algorithm which is a combination of the LZ77 and the Huffman coding algorithms. The change came due to the complexity of having to implement the gzip algorithm from scratch. The following will describe the experimental setup for the compression algorithms used. The compression algorithm experiment was set up in the following manner. The data from the csv files were used to test the compression algorithm and the way they were tested is by compressing the data from the csv file line by line instead of the whole file then have the same line decompressed so that it could be compared to the original data to ensure that there has been no data tampering. The experiment was tested on all the csv files provided so the experiment was run for 9 times so that means it covers enough ground for being viable given the data type.

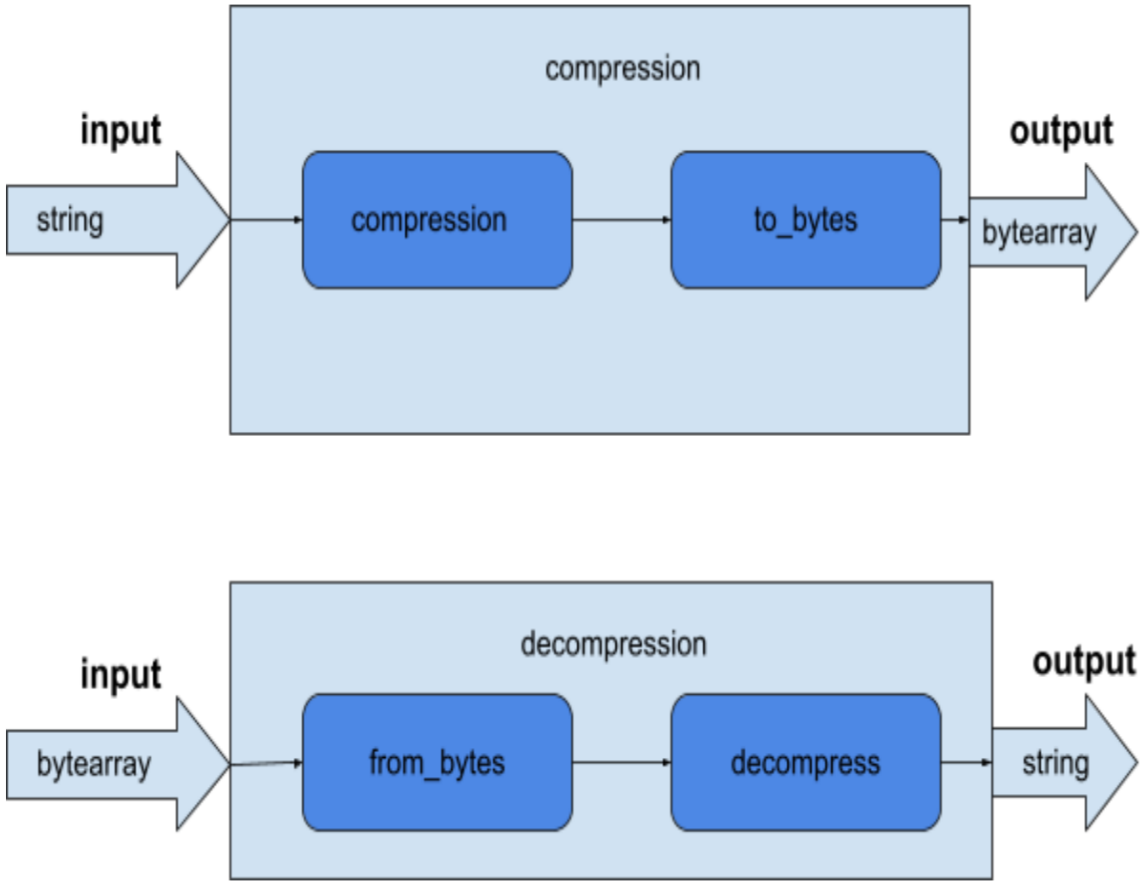


Figure 5 : Compression/Decompression sub-system

3.1.3.2. Encryption Block

Based on the above past papers cryptographic methods analysis and requirements, we will implement the AES algorithm to fulfill the associated specifications. The algorithm is widely used, efficient and feasible for the design.

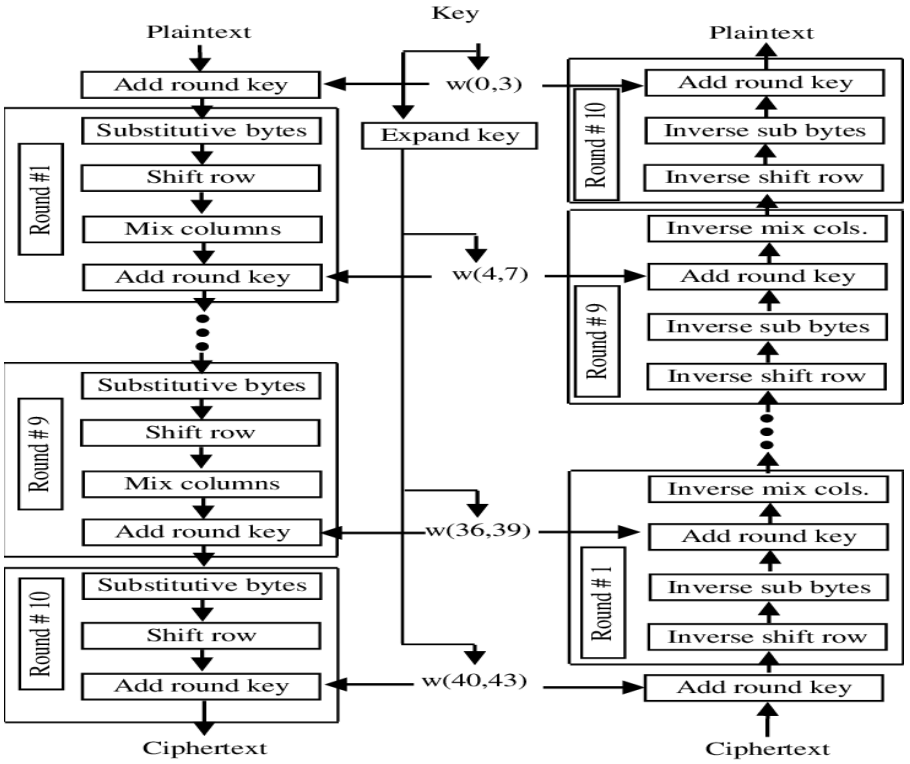


Figure 6 : Encryption/Decryption sub-system

3.1.4. UML Use Case Diagram

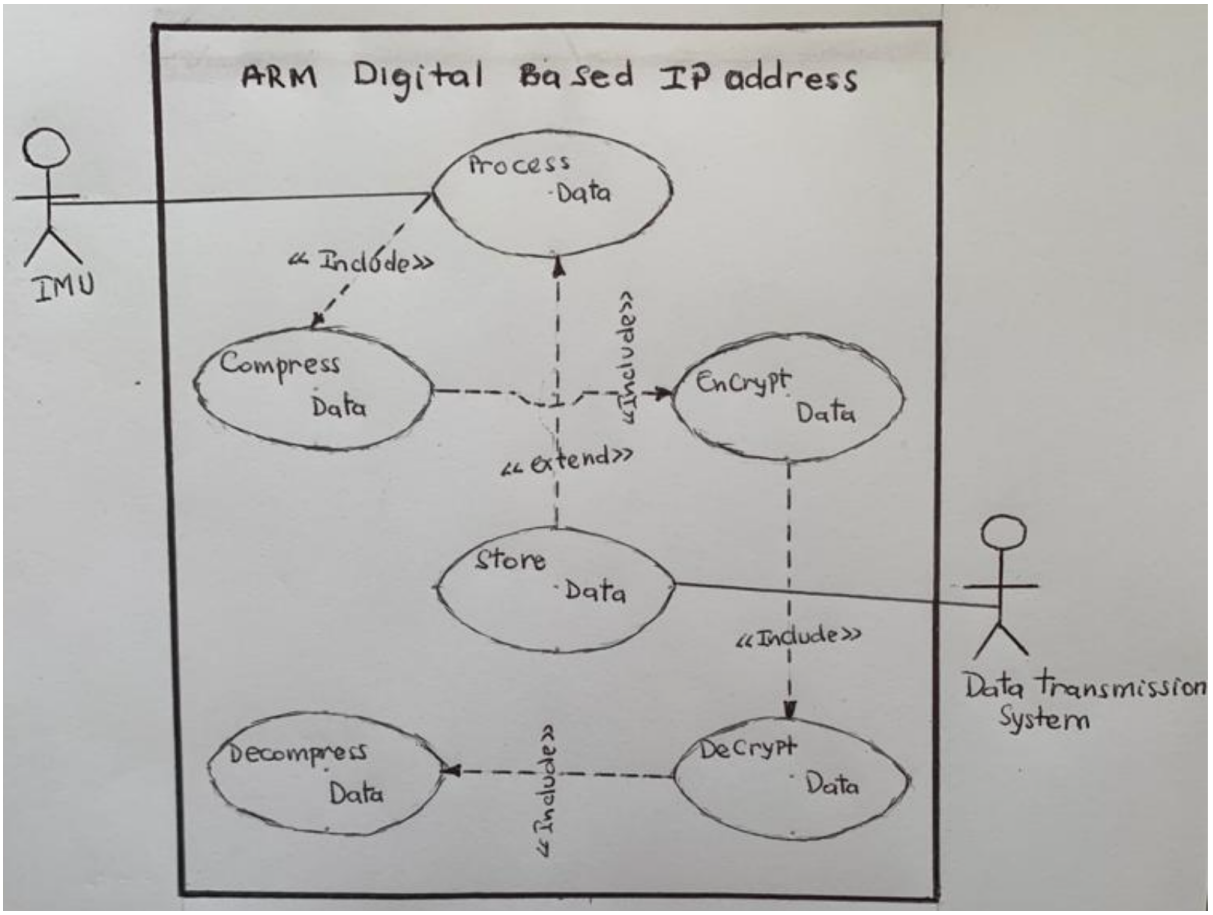


Figure 6 : UML Use Case Diagram

4. Validation using Simulated or Old Data

Simulation based validation

The need to perform simulation based validation is high because it allows you to see the flaws of the design and the system without it costing too much as compared when you are testing it using hardware because when a mistake has been made it might be costly depending on the mistake that has been committed. With simulated based validation you can even test extreme edge cases if they are extremely rare to ensure that your system will be able to take care of those cases.

List of steps

The steps taken to validate the simulated data is by creating sin, cos, tan, gaussian noise, all the mentioned trig functions superimposed with noise, and the csv files we were given for the project.

Experiment Setup

4.1.1. Experiments/Simulations of the overall functionality of the system

The whole system will have the data being generated by the IMU and the data that is generated by the IMU will be stored in memory where it will be passed to the compression subsystem where the data will be compressed and then the compressed data will be sent to the encryption subsystem where the compressed data will be encrypted and then it will come out as encrypted data. The encrypted data will then be unencrypted and then after being unencrypted it will be decompressed and then the decompressed data will be checked if it is exactly the same as the data that was sent so that it is confirmed that there was no data corruption during all the conversion that were occuring.

4.1.2. Experiments/Simulations of the compression and decompression blocks

The setup for requirement 1.10 the uncompressed data will be sent to be compressed and then after the compression it will be sent to the client to show what the data looks like when compressed and it will be decompressed on the client side so to check if there was no corruption of the data due to the compression.

The setup for requirement 1.3 is to have the compression algorithm continuously running where it will be continuously compressing data, while monitoring the CPU to see if the compression algorithm does utilize the CPU to 100%.

The setup for requirement 1.9 will have the uncompressed data be streamed to a file and the compressed data will be streamed to a file too, then at the end of the stream the 2 file sizes will be compared to ensure that the compressed file is less than the uncompressed file.

The setup for requirement 1.5 will be set up in the following manner, the data will be streamed to a file and those files will contain data at different frequencies to see if the lower 25% of the fourier coefficients are lost or not.

4.1.3. Experiments/Simulations of the encryption and decryption blocks

As mentioned in both the paper design section, we will use the Advanced Encryption Standard (AES) algorithm for the experiment setup to be run against our ATPs . The idea is to limit this simulation to the use of the AES private cryptosystem key of 128 bits length.

However, an extra section will be introduced to compare and understand key differences between the AES and algorithms such as the DES and RSA to be precise.
Since the security features of each algorithm as their strength against cryptographic attacks was already known and discussed in the paper design. The chosen factor here to determine the performance of each algorithm will be the speed to encrypt/decrypt data blocks of a couple of various sizes.
The setup for **requirement 1.14**, where the provision of confidentiality and data integrity to users should be guaranteed, will consist of representing the compressed data type (plaintext, ciphertext and key) the encryption and decryption blocks are supposed to get in and give out, as strings in matrices. Note that every bytes arrays or bytes strings will be decoded and processed as strings:

- **Plaintext Block Storage**

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

➢ Every *p* entry within the above matrix is a byte (8 bits). Therefore 16 entries x 8 bits *x* = 128 bits, which will be the size of a plaintext block.
➢ Every column represents a word (1 word is 32 bits long chunk or 4 bytes).
➢ Note that we are referring to a column by column representation.

- **Private Key Storage**

We will store the output, the intermediate result and the key as a matrix like this:

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

- **AES Diagram**

The security features of the AES algorithm, as part of of the setup for requirement 1.14, will be implemented as per the below diagram:

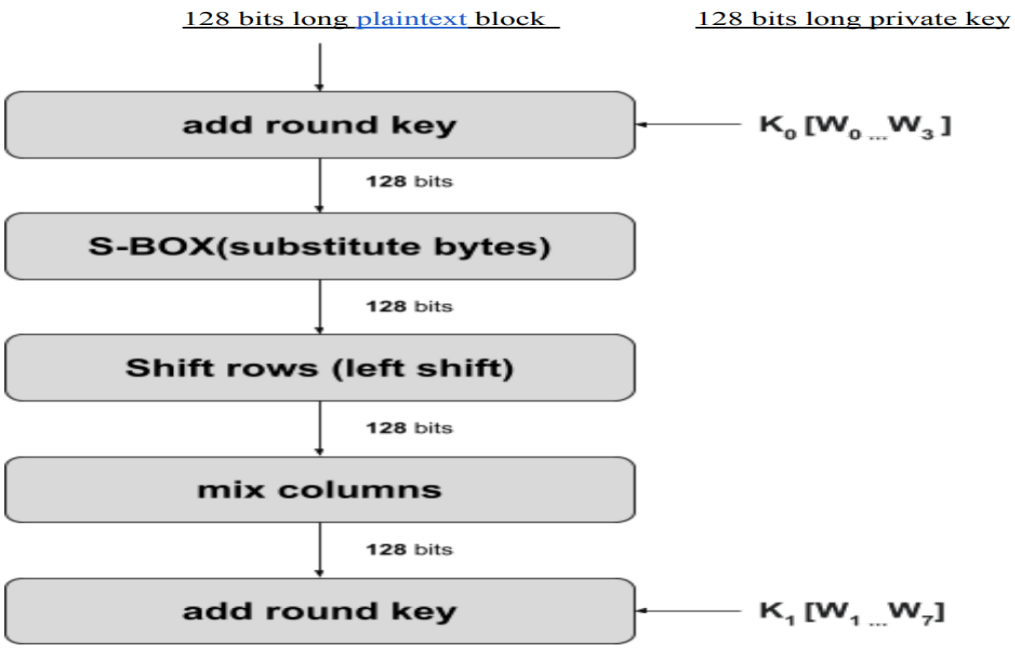


Figure 7: AES Diagram

Here is the elaboration of what each step in the setup will be doing:

➔ **Round Function**

○ In every round, a substitution will be made so the S-BOX then mix rows and columns and finally add round key operation.

○ In every round, a different subkey, generated from the private key, will be used.

○ In the last round, we won't use the mix columns operation

➔ **Add Round Key Operation**

This is going to be an XOR operation in which:

○ The input plaintext block is 128 bits long sequence

○ The private key is a 128 bits long binary sequence

We will just have to use bitwise XOR operation on a bit-by-bit basis. That is, the first bit of the plaintext is XORed with the first bit of the private key and so on.

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

The output will be 0 or 1 with 50% probability to ensure the ciphertext is approximately random

➔ **Substitute Bytes Operation (S-BOX)**

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

A single *p* entry (8 bits) is going to be the input of the s-box for a returned output of 8 bits. *x* We will consider all the 16 items in the matrix and for every item, we will apply the s-box.

0 1 0 1 1 1 0 0
Row Index *Column index*

Because we have 4 bits for the rows and 4 bits for the column, that's why this look-up (s-box) table should be a 16x16 table with values that are carefully chosen to be resistant to linear and differential crypto-analysis.

➔ **Shift Rows Operation (Circular Left Shift)**

Here, we have a matrix as an input again:

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

For row 1: We will use a circular left shift with 0 steps. Meaning no shifting required

For row 2: We will use a circular left shift with 1 step. That is, shifting the bytes in the row, one shift to the left with 8 bits.

For row 3: We will use a circular left shift with 2 steps. That is, shifting the bytes in the row, two shifts to the left with 16 bits

For row 4: We will use a circular left shift with 3 steps. That is, shifting the bytes in the row, three shifts to the left with 24 bits.

The resulting matrix, after shift rows operation, will look like this:

S_0	S_4	S_8	S_{12}
S_5	S_9	S_{13}	S_1
S_{10}	S_{14}	S_2	S_6
S_{15}	S_3	S_7	S_{11}

➔ **Mix Columns Operation**

Here, we are going to multiply the input matrix with another random predefined matrix.

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

2	**3**	**1**	**1**
1	**2**	**3**	**1**
1	**1**	**2**	**3**
3	**1**	**1**	**2**

As mentioned above, this step in AES will consist of a matrix-vector multiplication. That's taking the columns from the state-matrix and multiplying the predefined matrix with these vectors. After the complete mix-column operation, the transformed state matrix will look like this:

S_0'	S_4'	S_8'	S_{12}'
S_1'	S_5'	S_9'	S_{13}'
S_2'	S_6'	S_{10}'	S_{14}'
S_3'	S_7'	S_{11}'	S_{15}'

- **Subkeys Generation**

The setup for **requirements 1.12 and 1.13**, where the provision of a manageable encryption and decryption key size is highlighted, will consist of having the private key represented as a two dimensional array or matrix at the beginning of the algorithm. In this array or matrix, every entry represents one byte making the matrix a 128 bits long private key. The column of these private keys will be used to generate the given subkeys. These subkeys will be generated on a word by word (one column after the other) basis:

➔ Rotation operation will be applied. This is equivalent to a binary left shift but in this case we shift the bytes one step upwards in a circular manner.

➔ Then S-box will be used with the round function for every single byte:

- First 4 bits: row index in the lookup table
- Last 4 bits: column index in table

➔ The XOR operation will be used with previous words in the key and the values in the rcon table.

This process will be repeated for 10 rounds to generate 10 subkeys required for the number of 128 bits in the private key cryptosystem.

NB: The setup for **requirements 1.15,** to assure compatibility of the AES algorithm with the data from the ICM-20649 IMU (sensor)**,** will consist of ensuring the generated IMU data are in the same format as the csv file data used for experiment is the first progress report.

4.1.4. Results of the overall functionality of the system

The overall system had CPU utilization was on average at 24% so this means that the overall system wasn't power hungry due to the level of utilization that it achieved, if the utilization was on average closer to 90% then that would mean the overall system was consuming too much power.

4.1.5. Results of the compression and decompression blocks

When requirement 1.7 was being tested the results showed the CPU usage to have been 21-27% for the 10 different datasets that were used for the ATP. The results indicate that the algorithm used is not CPU intensive which in turn means that the CPU is not using so much power to process the data.

When requirement 1.9 and requirement 1.7 were being tested on the 10 various data sets the compression algorithm actually caused the data to increase instead of decrease; this could be caused by the fact that the compression is done on a line by line basis rather than the whole file. The whole file approach is not used because the runtime of the algorithm becomes far too long for it to be useful given the linear nature at which the data is read and how the window size of the compression algorithm is created. If the whole file was to be compressed as a whole then the window size would really struggle to process the data because it would instead turn into a factorial runtime algorithm. The data size becomes 3x larger than the data size. The other reason why the algorithm turns into a larger data set is because the algorithm is best used for words rather than numbers.

5. Validation using a different-IMU

The difference between the ICM-20649 IMU and the Sense Hat(B) IMU are the following:

- The Sense Hat (B) uses an I2C interface only while the ICM-20649 uses both I2C and SPI.
- The Sense Hat (B) can sense pressure, humidity, and color while the ICM-20649 cannot sense all of those.
- The Sense Hat (B) works with 3.3 volts while the ICM-20649 works with 1.71 volts to 3.6 volts.

- The Sense Hat (B) is 65 mm x 30.5 mm while the ICM-20649 is 3 mm x 3 mm x 0.9 mm.
- The gyroscope on the Sense Hat (B) has up until +/- 2000 dps (degree per second) while the ICM-20649 has up until +/- 4000 dps (degree per second). The ICM-20649 has a programmable low-pass filter.
- The accelerometer on the Sense Hat (B) has up until +/- 16g while the ICM-20649 has up until +/-30g, also included is a wake-on-motion interrupt for low power operations.
- There is a magnetometer on the Sense Hat (B) while there isn't one in the ICM-20649, it can only be attached externally.
- The Sense Hat (B) has a 12-bit ADC while the ICM-20649 has a 16-bit ADC.

The differences are listed

The list of steps taken to make sure the IMU is similar to the IMU used in the actual buoy are as follows:

- The IMU will forever be in motion due to it being in the waves; this means when we test the IMU we will ensure that the motion that would be expected in the waves is done on the IMU.
- We have ensured that the data that is being handled is not written instead it is handled in memory due to the fact that the actual buoy has limited space and it will need to be transmitting the data as often as possible.
- We have handled everything when it comes to the IMU with the assumption that manual intervention will not occur so we have ensured that the code is run in a way that is inline with the environment as in there is no displaying of information since there will be no monitor to display it to.

List of validation tests to make sure the IMU is working as expected:

- The motion that will be used to test the IMU will vary from the gentle movements that the IMU would experience when the buoy is in gentle waves and then the movement will be changed to rapidly moving to test for movement for when it is experiencing very rough weather. This will be testing the accelerometer of the sensor.
- The gyroscope will be tested by having the IMU put in various positions to see if the gyroscope is actually working and the program will be started in various positions in order to see if the gyroscope functions as expected and it doesn't lose it's tilt/rotation ability to sense.

- The magnetometer unfortunately cannot be tested due to the lack of a magnet.

The data used for this section was similar to the one that we were provided with from the csv files however it was only limited to the following parameters:

- Accelerometer all 3-axis.
- Gyroscope all 3-axis.

The overall results.

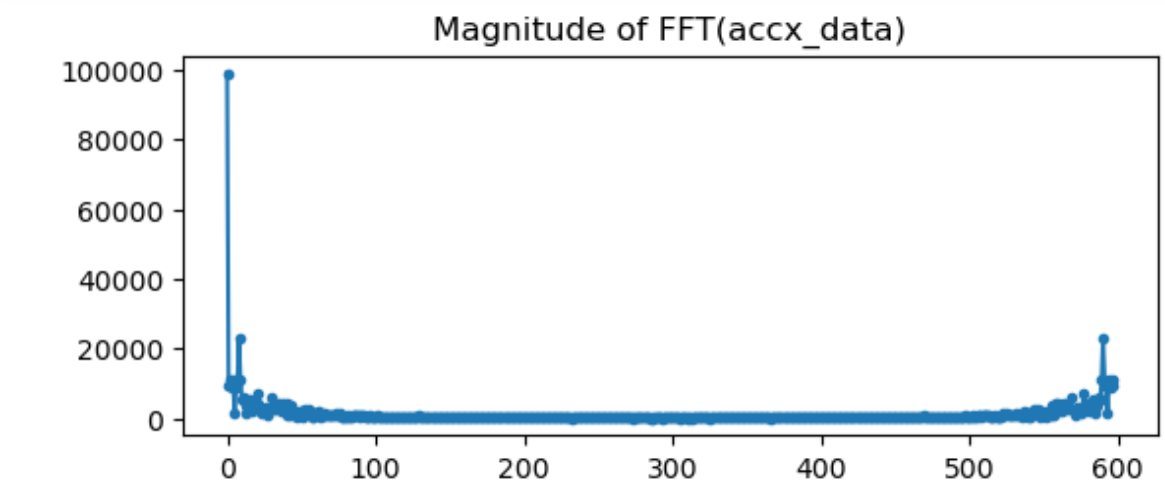
The overall results have remained the same as the results achieved using the simulated data.

The compression and decompression results.

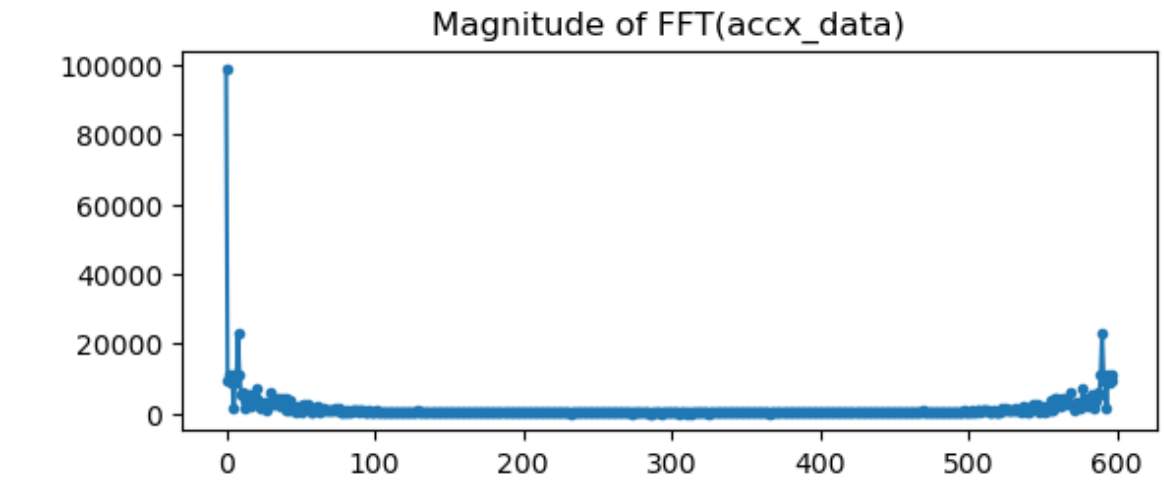
All the other requirements that were tested using the simulated data from the IMU produce the same results.

The Fourier Transform Data for Requirement 1.2

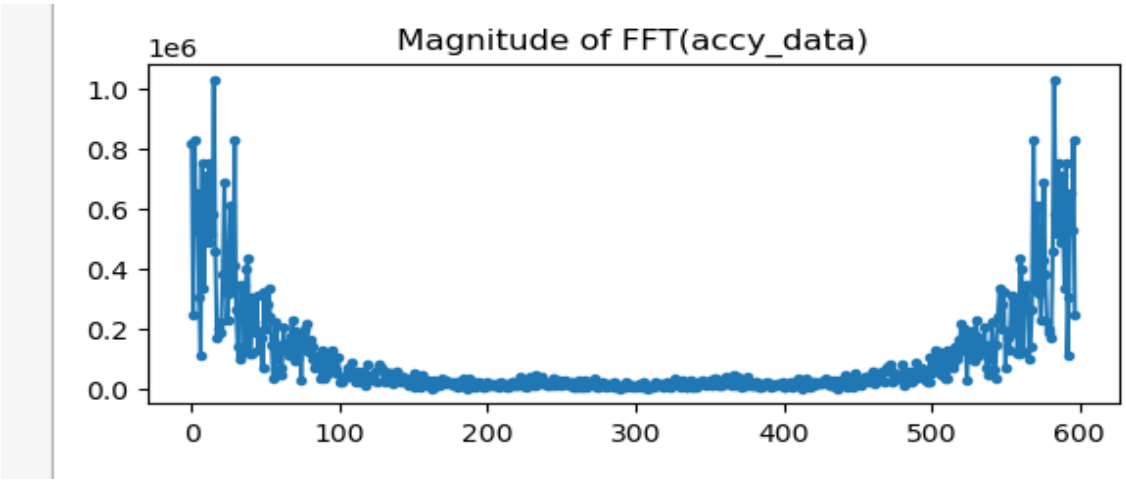
The following graph shows the accx fourier transform before the compression algorithm:



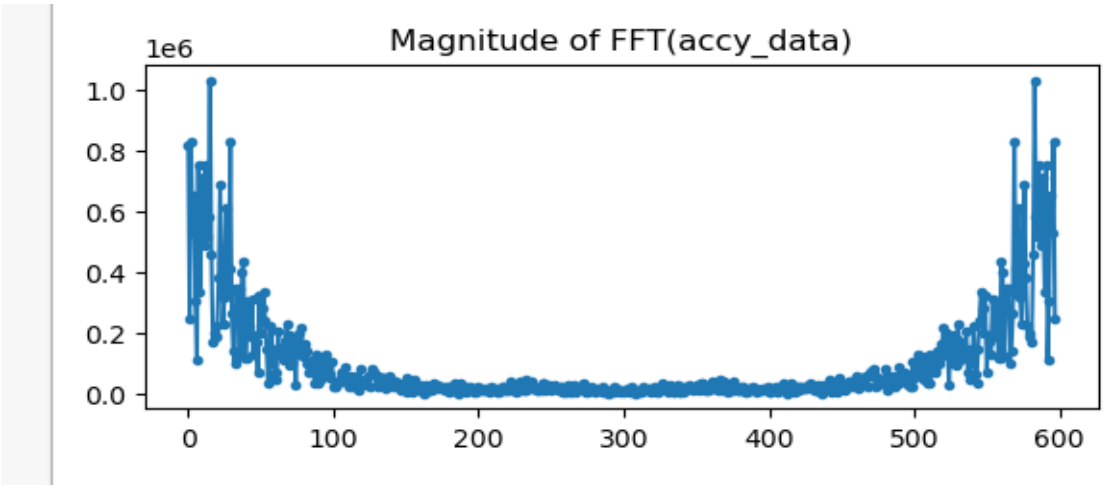
The following graph shows the accx fourier transform after the decompression algorithm:



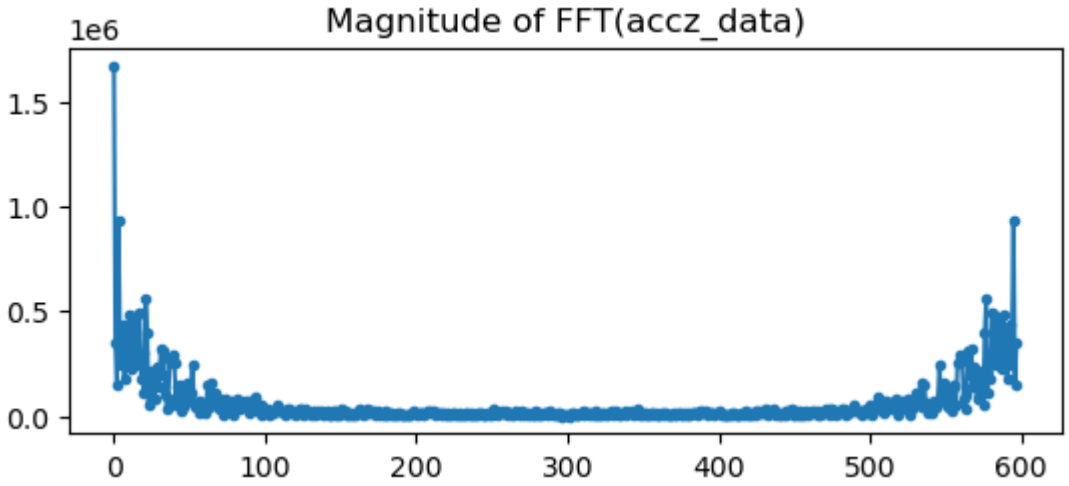
The following graph shows the accy fourier transform before the compression algorithm:



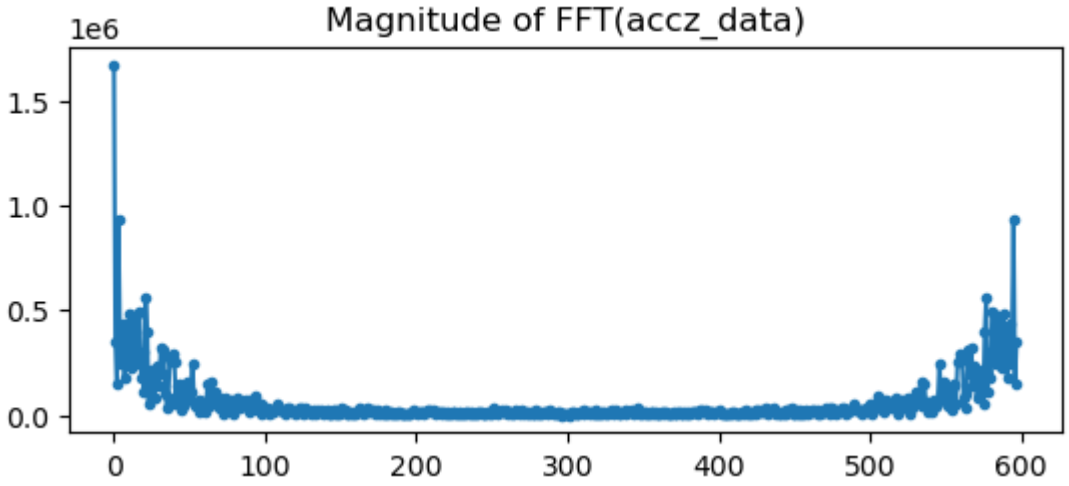
The following graph shows the accy fourier transform after the decompression algorithm:



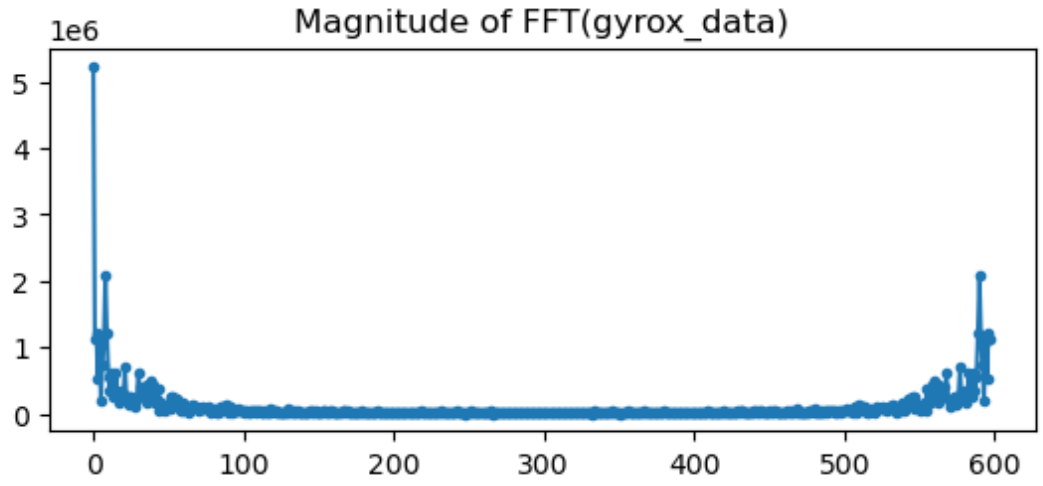
The following graph shows the accz fourier transform before the compression algorithm:



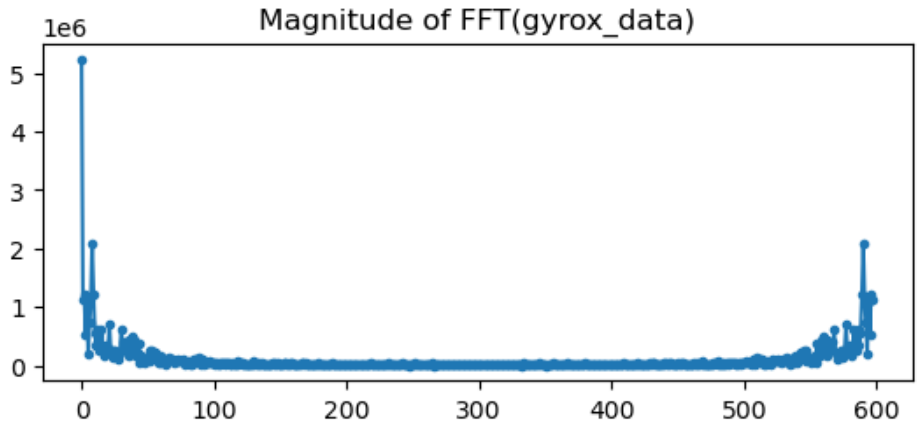
The following graph shows the accz fourier transform after the decompression algorithm:



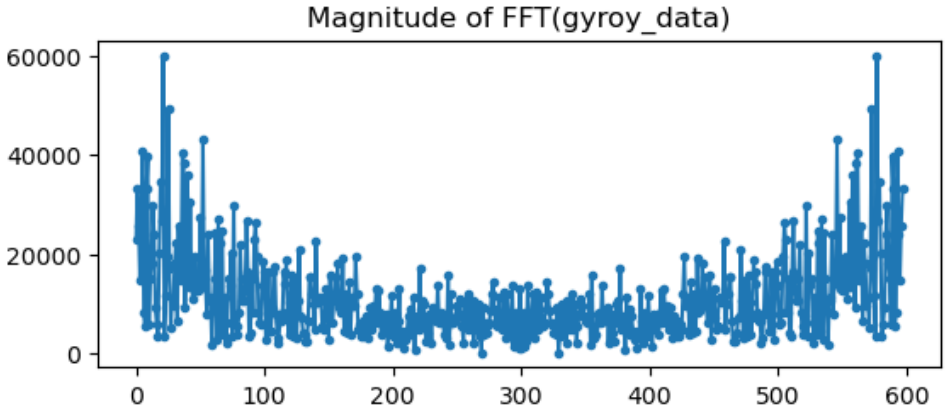
The following graph shows the gyrox fourier transform before the compression algorithm:



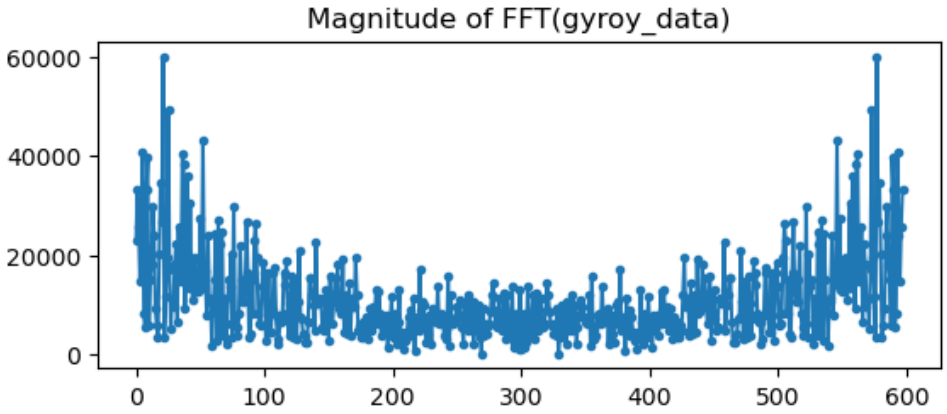
The following graph shows the gyrox fourier transform after the decompression algorithm:



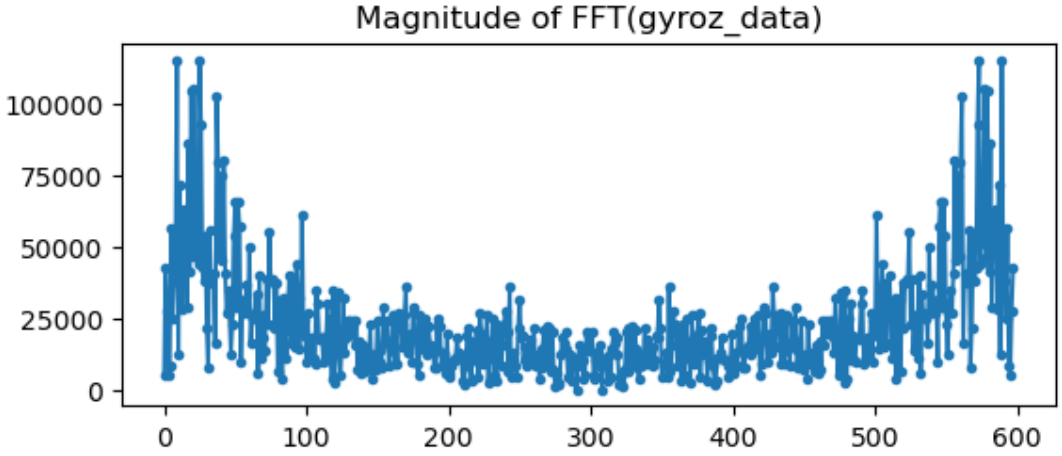
The following graph shows the gyroy fourier transform before the compression algorithm:



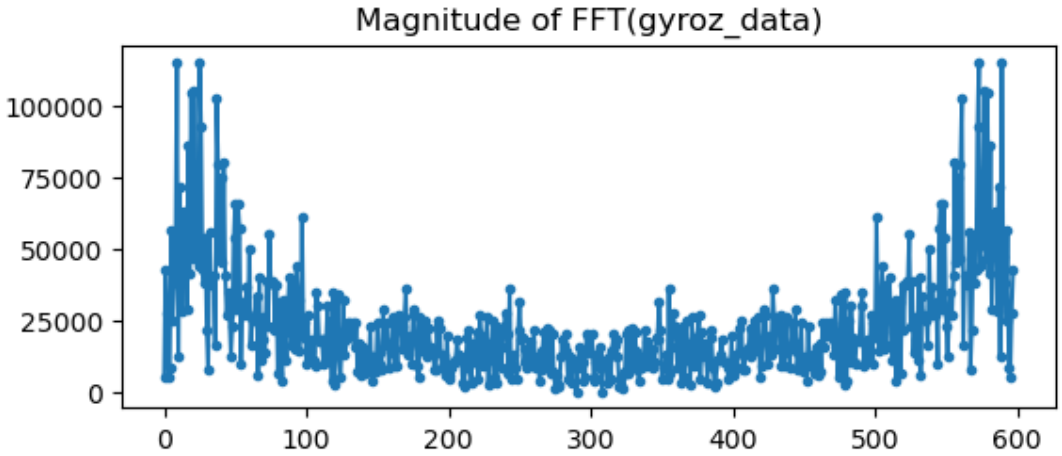
The following graph shows the gyroy fourier transform after the decompression algorithm:



The following graph shows the gyroz fourier transform before the compression algorithm:



The following graph shows the gyroz fourier transform after the decompression algorithm:



When requirement 1.8, the compression and decompression of the 10 different sets of data did not corrupt the data because the data that came in was the data that came out.

The encryption and decryption results

When requirement 1.14 was tested, the result showed that the AES algorithm was implemented to compile using the Cipher Block Chaining that uses a mechanism that causes the decryption of a block cipher text to depend on all of the preceding ciphertext blocks to avoid information leaking.

Padding was taken into account considering the fact that the AES algorithm will be using 128 bits long input. With padding in place, we were able to append some extra bits to the plaintext to take care of the case when the input plaintext from the IMU was not divisible by 128. As a result, we were able to split the plaintext into 128 bits long chunks.

When requirements 1.12 and 1.13 were tested, the result showed that we made use of the initialisation vector to ensure that even if we use the same key and same plaintext, the encrypted results are different. This is shown in simulated figure 8 below. During the implementation of the decryption function, we used the same initialisation vector used for encryption, to ensure we get the original plaintext.

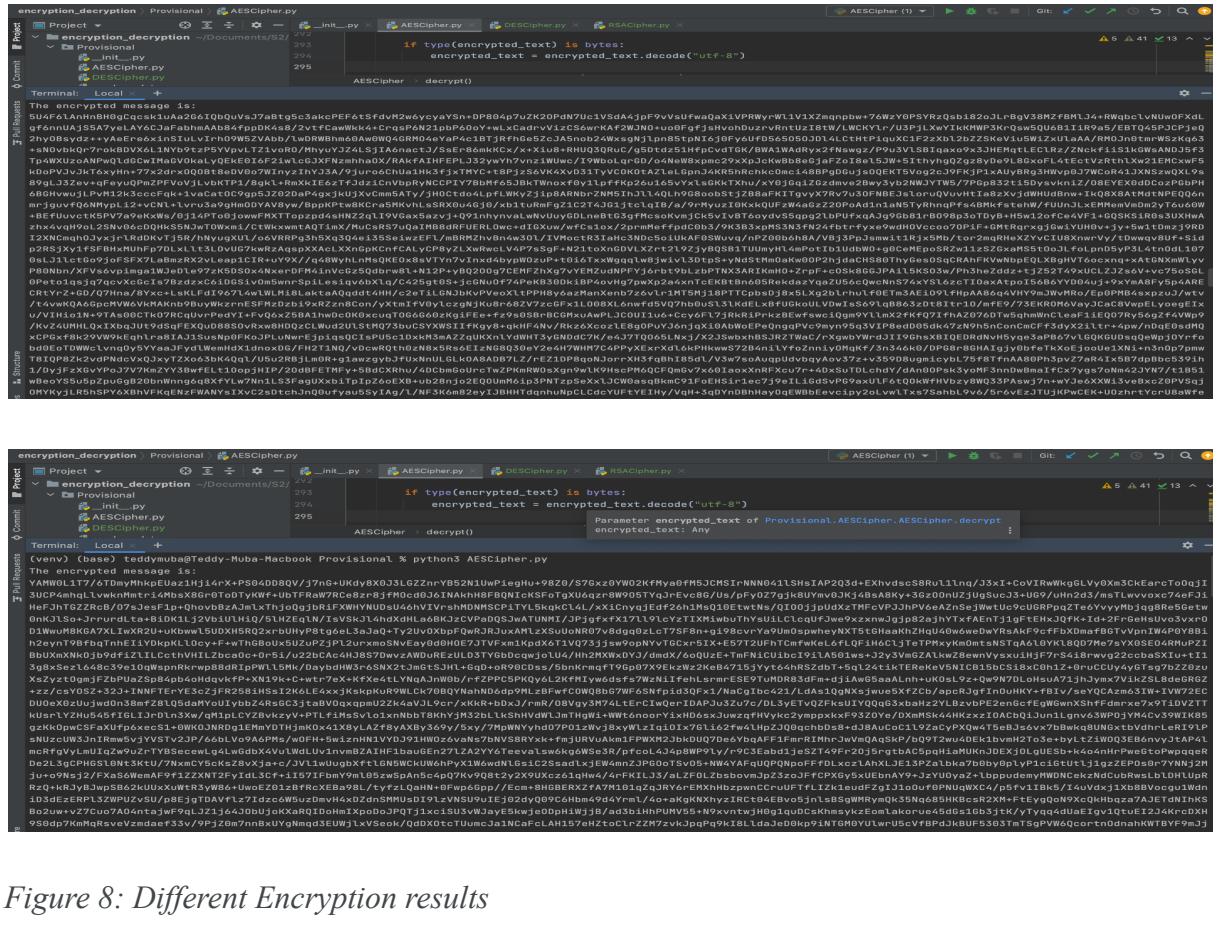


Figure 8: Different Encryption results

6. Consolidation of ATPs and Future Plan

Compression & Decompression

Requirements	Specifications	Acceptance Test Procedure	Pass/Fail
R1.3: The compression of the data should not be CPU intensive given any data size.	**S1.3:** The compression algorithm has to have an O(n) time complexity	This will be tested using 10 random data sets of various sizes to ensure the time complexity is kept.	Pass
R1.9: The compression data should be smaller than the original data	**S1.8:**The compressed output has to be less than the original.	This will be tested using 10 various data sizes and it will be checked if the data size after compression is smaller than the original and that the data when decompressed is the exact same size as the original.	Fail, the ATP has not been met due to the design structure of the compression algorithm; it works well with text rather than numbers so then the algorithm would have to be redesigned using a library rather than coding it from scratch.
R1.2: The data read by the Oceanographers should at least have the lower 25% of the fourier coefficients.	**S1.1:**At least 25% of the lower Fourier coefficients need to be in the data	This can be tested by running the solution with 10 various sizes of random data and comparing the Fourier coefficients of the original and the compressed version.	Pass
R1.10: The compressed data should be decompressed on the other side.	**S1.8:**There should be 0% of data corruption	This will be tested by comparing the random data contents and the contents of the compressed version of the random data and confirm that all values in the compressed version do exist in the uncompressed version.	Pass

		This will be ran over 10 samples	
R1.5: The sample rate should not be excessive.	**S1.5:**The sample rate should be at 800 Hz	Will confirm with the Fourier coefficients if it has been sampled at 1 kHz and this will be tested for 10 random samples	Pass
R1.6: The data doesn't need to be sent in real time. There can be delays with it being sent.	**S1.6:**Raspberry Pi should be able to handle FIFO of 4kB	The Raspberry pi will be tested using 10 random samples with data that is greater than 4kB to test if the Raspberry pi can handle the speed.	Pass

5.2 Encryption & Decryption

Requirements	Specifications	Acceptance Test Procedure	Test Execution	Pass/Fail
R1.12: Provision of a manageable encryption key size.	**S1.12:** AES should have a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits	This can be tested using the numbers of cycles of repetition as follows: ● 10 cycles of repetition for 128-bit keys. ● 12 cycles of repetition for 192-bit keys. ● 14 cycles of repetition for 256-bit keys.	10 rounds (iterations) implemented, executed and tested to generate 10 subkeys required for the number of 128 bits in the private key cryptosystem	**Pass**
R1.13: Provision of a manageable decryption key. size. The data receiver should be able to view the original	**S1.13:** AES should have a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits	This can be tested using a set of reverse rounds applied to transform ciphertext back into the original plaintext.	10 rounds (iterations) implemented, executed and tested to generate 10 subkeys required for the number of 128 bits in the private key cryptosystem	**Pass**

data by using decryption technique.				
R1.14: Provision of confidentiality and data integrity to users.	**S1.14:** ● To encrypt, an implemented algorithm has to run and encrypt the plaintext using a generated subkey. ● Running the algorithm multiple times should generate different encrypted messages. ● To decrypt, an algorithm has to run and decrypt the decrypted text using the same subkey generated above.	This can be tested when going through the following steps: ● Powering ON the RaspberryPI and LCD display. ● Starting the OS by using start_x command. ● Opening terminal in the RaspberryPI desktop. ● Running the file in the terminal of the RaspberryPI desktop. ● Running the code and the algorithm will run and use the generated subkey to encrypt the file. ● The algorithm will continue running and use the generated subkey to decrypt the file.	● Running the implemented algorithm encrypted the plaintext using a generated subkey. ● Running the algorithm multiple times generated different encrypted messages due to the fact that at every run, a new encryption subkey was generated. ● The algorithm was run to decrypt the ciphertext using the same above encrypted key	**Pass**
R1.15: Assuring compatibility with the ICM-20649 IMU (sensor).	**S1.15:** The RaspberryPI specifications should be aligned to this requirement	This can be tested by: ● Initially enabling the ARM board by giving the power supply. ● Connecting keyboard, mouse, lcd display to the ARM. ● Opening Qt/or any other software in the LCD display. ● Building, compiling, running the code in Qt/or any other software ● Selecting cover file, secret file from the SDcard	The RaspberryPi was tested using 10 random samples with data that is greater than 4kB to test if the Raspberry pi can handle the speed.	**Pass**

7. Conclusion

For the compression side the issue with having the compression algorithm creating larger files rather than making them smaller is of great concern due to the nature of the project and the results that were seeked, because this would make compressing the data more expensive when compared to sending the data without compression. The additional data was caused by having to code the compression algorithm from scratch rather than using a library because when the code was tested using a library it ensured that the compressed data was less than the original size of the data. Having said that, the gzip algorithm is still the better choice due to its low runtime and high compression ratio for that compression time taken.

As a summary on the encryption side, each of the considered algorithms appeared to offer adequate security, and each offered a considerable number of advantages. Any of the studied algorithms (DES, RSA, etc.) could serve admirably as the AES. However, each algorithm also has one or more areas where it does not fare quite as well as some other algorithm. We selected AES as the proposed algorithm at the end of a very long and complex evaluation process. During the evaluation, we analyzed some public comments, papers and reports around the topic. We judged AES to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time being excellent, and its key agility being good, AES very low memory requirements made it very well suited for the encryption of the maximum of the compressed amount of byte_arrays we were expecting, in which it also demonstrated excellent performance. Its flexibility in terms of block and key sizes could allow this algorithm to accommodate alterations in the number of rounds, although we only limited our experiment to a 128 bits (10 rounds) long compressed plaintext. When considered together, AES 'combination of security, performance, efficiency, implementability, and flexibility made it an appropriate selection for the design of an ARM based digital IP using a Raspberry-Pi to encrypt an IMU environmental compressed data.

References

1. Elminaam, Diaa Salama Abd, Abdual Kader, Hatem Mohamed & Hadhoud, Mohiy Mohamed. “Evaluating The Performance of Symmetric Encryption Algorithms”. International Journal of Network Security, Vol.10, No.3, May 2010, pp. 216.

2. Padmapriya, Dr.A, Subhasri, P. “Cloud Computing: Security Challenges & Encryption Practices”. International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X, Volume 3, Issue 3, March 2013, pp. 257.

3. Advanced Encryption Standard (AES). FIPS. November 23, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (accessed March, 15, 2010).