
Submission 4: Final Report

Course Code: EEE3097S
Authors: Bradley Carthew, Nathanael Thomas
Student Numbers: CRTBRA002, THMNAT011
Date: 31 October 2021

1. Admin Documents

1.1. Contributions

Contributor	Tasks	Sections	Page Numbers
Nathanael Thomas, THMNAT011	<ul style="list-style-type: none">• Link GitHub repository• Make Timeline• Compile the specifications• Contributed to the paper design• Need for simulation based validation• Steps taken to execute the simulations• Explanation of the encryption and decryption of the old/simulated data• Compilation of the results obtained from the old data from IMU and execution of the algorithms on the entire system• Encryption and decryption algorithm execution for the data obtained from the old IMU data• Explaining the process taken to execute the compression and encryption as well as decryption and decompression of the data in relation to system as a whole• Encryption and decryption algorithm execution for the data obtained from the sense hat B• Explanation of the encryption of the data obtained from the senseHat B• Change of the specifications• Future plans around the project• Contribute to the conclusion• Referencing compilation	<ul style="list-style-type: none">1.31.42.33.4.14.24.3.34.4.14.4.35.3.15.3.35.4.36.26.37.8.	<ul style="list-style-type: none">334-77-111111-121516-1720-2121-232327282929-3030-31
Bradley Carthew, CRTBRA002	<ul style="list-style-type: none">• Compile contributions table• Get screenshot of Trello page• Discuss requirements• Generate list of requirements• Contribute to paper design• Compile experimental setup for system and compression in validation using simulated or old data section (4.).• Compile results for compression in validation using simulated or old data section (4.)	<ul style="list-style-type: none">1.11.22.12.23.4.3.14.3.24.4.2	<ul style="list-style-type: none">1- 2233-47-1112-131418-19

	<ul style="list-style-type: none"> Need for hardware based validation Steps taken to execute hardware based validation Compile experimental setup for compression in validation using different IMU section (5.). Compile results for system and compression in validation using different IMU section (5.) Analyse the ATPs Contribute to the conclusion Compile list of references 	5.1 5.2 5.3.2 5.4.1 5.4.2 6.1 7. 8.	21 21 23 23-27 27 28 29-30 30-31

Table 1: A table showing the contribution of each of the team members.

1.2. Project Management Tool

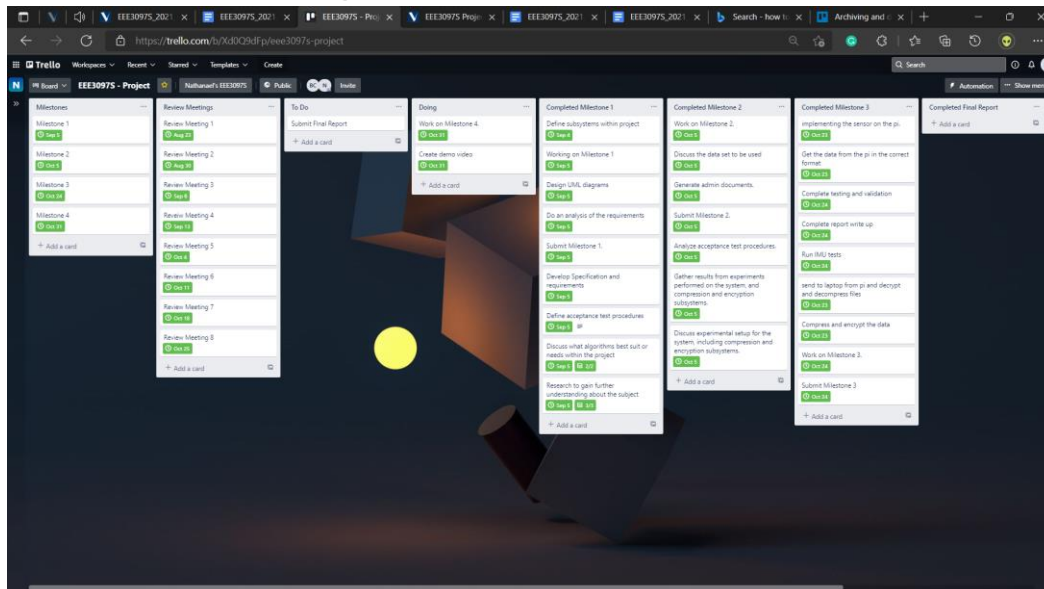


Figure 1: Snapshot of our Trello page.

Link to Trello: <https://trello.com/b/Xd0Q9dFp/eee3097s-project>

1.3. Link to our GitHub Repository

https://github.com/bradcarthew1/THMNAT011_CRTBRA002_EEE3097S

1.4. Timeline

Below is a basic Timeline demonstrating our progress throughout the development and implementation stages of our project.

PROJECT NAME		PROJECT DURATION	PROJECT START DATE	PROJECT END DATE
IMU simulation		76	16/8/2021	31/10/2021

Task ID	Task Description	Task Duration	Start Date	End Date	Aug,16,2021	Aug,22,2021	Aug,23,2021	Aug,29,2021	Aug,30,2021	Sep,5,2021	Sep,6,2021	Sep,12,2021	Sep,13,2021	Sep,19,2021	Sep,20,2021	Sep,26,2021	Sep,27,2021	Oct,3,2021	Oct,4,2021	Oct,7,2021	Oct,11,2021	Oct,17,2021	Oct,18,2021	Oct,24,2021	Oct,25,2021	Oct,31,2021	Nov,1,2021
1	Milestone 1(Paper Design)	20	Aug,16,2021	Sep,05,2021																							
2	Milestone 2 (progress report 1)	30	Sep,05,2021	Oct,05,2021																							
3	Milestone 3 (progress report 2)	19	Oct,05,2021	Oct,24,2021																							
4	Milestone 4 (final report)	7	Oct,24,2021	Oct,31,2021																							
5	Demo video	7	Oct,24,2021	Oct,31,2021																							

Figure 2: Timeline of our progress taken from our Trello account.

The only delay was during milestone 3 when waiting to receive the sense hat B. This pushed back progress a little. But had not exponential effects on completing the milestone before the due date.

See the link to our Trello page for a more in-depth look at our milestones' progress.

2. Requirement Analysis

2.1. Discussion of Requirements

The initial requirements given to us at the start of the project, in the problem statement, are listed below:

- Req1: The IMU to be used is ICM-20649. However, we shall not provide you these IMUs. So you shall need to find out ways to design your IP without the real hardware!
- Req2: Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data. Make sure that your compression satisfies this.
- Req3: 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). Try to minimize the computation required for your IP.

2.2. Requirements

Below is the list of the derived requirements, they have been split up into hardware and software subsystems for easy reading.

Hardware Subsystem Requirements:

- An alternative processing unit must be used to implement our design, because the IMU on the SHARC Buoy will not be made available
- The processing unit must be able to generate datasets using on-board sensors
- The processing unit must have a power source
- The processing unit must be able to connect to a network
- The processing unit must be able to implement and execute compression and encryption algorithms

Software Subsystem Requirements

Compression

- Data after decompression must be the same as the original dataset
- Data must be compressed to a size smaller than that of the original file's size
- Compression algorithm must include the compression and decompression of datasets
- Compression must not be strenuous on the processing unit
- Compression must execute quickly
- Compression must not be too complex
- At least 25% of the Fourier coefficients must be accessible

Encryption

- Data outputted should be the same as the data that was inputted
- Encryption algorithm must include the encoding and decoding of dataset
- Encryption should be fast
- Encryption should not be strenuous on the processing unit
- Must be able to encrypt large sets of data
- The encryption algorithm must not be complex
- The dataset must be encrypted so as to eradicate data leaks

2.3. Specifications

Below is a list of desired specifications derived from the requirements (above, in Section 2.2). The specifications are split up into hardware subsystem and software subsystem specifications for easy reading. For each specification, an acceptance test procedure was developed so as to check that the specifications are met.

Hardware Subsystem Specifications

Requirement	Specification	Acceptance Test Procedure
An alternative processing unit must be used to implement our design, because the IMU on the SHARC Buoy will not be made available	Make use of a Raspberry Pi Zero W for processing and attach the Sense-HAT B to provide sensor functionality.	

The processing unit must be able to generate datasets using on-board sensors	On-board sensors must measure yaw, pitch, roll, magnetic fields (in x, y and z directions), gyroscopic movement (in x, y and z directions) and acceleration (in x, y and z directions).	Move the Sense-HAT B around in different directions and check to see that the yaw, pitch, roll, gyroscope and acceleration values are changing. Bring a magnet close to the Sense-HAT B to check that the magnetometer is functioning correctly.
The processing unit must have a power source	Make use of a USB to USB mini for connection from a laptop to Raspberry Pi Zero W to provide power.	Check to see that the green light on the Raspberry Pi Zero W and the red light on the Sense-HAT B turn on.
The processing unit must be able to connect to a network	Connect the Raspberry Pi Zero W to a laptop hotspot (WiFi adapter 802.11a/b/g wireless mode must be set to 2.4 GHz 802.11g).	Check to see that the device has been added to a list of devices connected to the laptops mobile hotspot.
The processing unit must be able to implement and execute compression and encryption algorithms.	The Raspberry Pi Zero W must be able to implement and execute gzip compression and AES encryption algorithms.	Compile and run the AES encryption and gzip compression algorithms on different datasets. Check that the files contents have been altered.

Table 2: List of requirements, specifications and acceptance test procedures for the hardware subsystem.

Software Subsystem Specifications

Compression

Requirement	Specification	Acceptance Test Procedure
Data after decompression must be the same as the original dataset	The contents of the file, after executing the gunzip algorithm, must be the same as the contents of the original file.	Use the windows powershell to run a bash script to check that the contents of the two files are the same. A message must be displayed indicating whether the file contents are the same or not the same.
Data must be compressed to a size smaller than that of the original file's size	The file size after executing the gzip algorithm must be to 50% smaller than the original file size.	Use the "ls -lh" command in the terminal to show the file size after compression and compare this to the original file size.

Compression algorithm must include the compression and decompression of datasets	Make use of the gunzip and gzip commands to decompress and compress data, respectively.	Compile and run the gzip compression algorithms on different datasets. Check that the files contents have been altered.
Compression must not be strenuous on the processing unit	The average CPU usage after running the gzip (or gunzip) algorithm must not exceed 50%.	Use the "sudo iostat -c 2 2" command in the terminal to show the average CPU usage before and after compression and decompression. An average CPU usage spike over 50% indicates that the algorithm is too power hungry, and anything below that indicates that the CPU usage is within valid bounds.
Compression must execute quickly	The gzip (or gunzip) algorithm execution time must not exceed 3 seconds.	Use a timer on a smartphone to check that after running the algorithm the execution time remains under the 3 second threshold.
Compression must not be too complex.	Make use of the gzip compression algorithm.	Implement different compression algorithms and run tests to see which algorithm executes faster.
At least 25% of the Fourier coefficients must be accessible.	At least 25% of the Fourier coefficients of the dataset must be made available after executing the gzip and gunzip compression commands.	Use the windows powershell to run a bash script to check that the contents of the two files are the same. If they are identical this indicates that 100% of the Fourier coefficients are available for processing.

Table 3: List of requirements, specifications and acceptance test procedures for the compression software subsystem.

Encryption

Requirement	Specification	Acceptance Test Procedure
Data outputted should be the same as the data that was inputted	The contents of the file, after executing the AES decryption algorithm, must be the same as the contents of the original file.	Use the windows powershell to run a bash script to check that the contents of the two files are the same. A message must be displayed indicating whether the file contents are the same or not the same.

Encryption algorithm must include the encoding and decoding of dataset	Make use of the AES encryption and AES decryption algorithms (Code Cracker algorithm) to encode and decode data respectively.	Compile and run the AES encryption and decryption algorithms on different datasets. Check that the files contents have been altered.
Encryption should be fast	The AES decryption and encryption algorithm execution time must not exceed 3 seconds.	Use a timer on a smartphone to check that after running the algorithm the execution time remains under the 3 second threshold.
Encryption should not be strenuous on the processing unit	The average CPU usage after running the AES encryption and decryption algorithm must not exceed 50%.	Use the “sudo iostat -c 2 2” command in the terminal to show the average CPU usage before and after encryption and decryption. An average CPU usage spike over 50% indicates that the algorithm is too power hungry, and anything below that indicates that the CPU usage is within valid bounds.
Must be able to encrypt large sets of data	The AES algorithm must be able to encrypt and decrypt a file containing around 15000 samples.	Run the encryption and decryption algorithm on a file with 15000 samples, check to see that the contents have been altered and that a message is displayed saying the file has been encrypted or decrypted successfully.
The encryption algorithm must not be complex	Make use of an AES encryption algorithm implemented using C++.	Implement the algorithm using different coding languages and run tests to see which language executes faster.

Table 4: List of requirements, specifications and acceptance test procedures for the encryption software subsystem.

3. Paper Design

3.1. Comparison of Compression and Encryption Algorithms

Compression

Compression is the process of reducing the size of data for the purposes of saving bandwidth, distributing data and saving space. [6]

There are two main methods used to compress data:

Lossy Compression [8]

Lossy compression involves reducing the size of data by getting rid of unnecessary or less relevant information. The amount of data is greatly reduced resulting in an improved data compression ratio. [6]

Advantages [7]

- File size is greatly reduced

Disadvantages [7]

- Is irreversible – original data cannot be completely restored
- Quality of the image is greatly reduced with a higher ratio of compression

Examples of Algorithms

- Discrete Wavelet Transform (DWT) - involves a Fourier analysis and decomposes a signal (set of data) into sinusoidal basis functions of different frequencies.
- Discrete Cosine Transform (DST) - decorrelates data, encodes each transform coefficient independently without losing compression efficiency. [9]

Lossless Compression [8]

Lossless compression involves reducing the size of data without any loss of data and is done by removing unnecessary metadata. [7]

Advantages [7]

- No loss in the quality of the data
- File size is reduced slightly

Disadvantages [7]

- Results in larger files than lossy compression

Examples of Algorithms

- Arithmetic Coding - a type of entropy encoding technique, that takes a file composed of symbols (eight-bit characters) and converts these symbols to a floating-point number that is greater than or equal to zero and less than one. [10]
- Huffman Coding – involves assigning a variable-length code to different input characters, this code length is related to how frequently characters are used. [11]
- Lempel Ziv Welch (LZW) - reads a sequence of symbols, groups the symbols into strings and converts the strings into codes. [11]

Encryption

Encryption can be defined as the process of converting readable data into encoded format. Encryption is used to provide privacy to the data as the encrypted data needs a personalised key either public or private or the use of a cypher depending of what was set/chosen by the person who had encrypted the data. [5]

There are two main types of encryption algorithms:

Symmetric Encryption

Makes use of a single key to encrypt and decrypt data.

Advantages [5]

- Simpler than asymmetric encryption
- Faster performance
- Less power used in computation

- Key length is smaller of size 128-256-bit length

Disadvantages [5]

- As there is only one key it needs to be kept secret
- Less private compared to asymmetric encryption

Three examples of algorithms [12]

- Two-fish – can encrypt a key of bits of length 256 bits. Only one key is required. Fastest symmetric technique and is openly available.
- AES – Advanced Encryption Standard, includes three block cyphers AES – 128,192,256. Depending on which cypher block is used, the cypher will encrypt and decrypt using cryptographic keys of the size of 128,192 or 256 bits, in blocks of 128 bits.
- DES- Data Encryption Standard it uses blocks of plaintext of a specific size 64-bits and will return an encrypted block of the same size.

Asymmetric Encryption

Defined as the encryption and decryption of data with the use of key pairs, known as public and private keys. Even though asymmetric encryption offers greater security through authentication. The downfall is that is easy to go in one direction (encrypt) but harder to retrieve the original [5]

Advantages [5]

- Provides better security then symmetric encryption
- Authentication, the data is only seen by the person who is supposed to see it

Disadvantages [5]

- Can be a very slow process
- High risks involved if one were to lose their private key. No longer able to decrypt the messages
- If anyone else were to know your private key they have access to all your messages, security breach
- Difficult to retrieve original data

Three examples of algorithms [5]

- RSA - makes use of prime factorisation. By multiplying two large prime number and to decrypt will find the original prime number.
- Diffie-Hellman - Allows for two parties to with no prior knowledge to have a shared key established between them over an insecure channel
 - ECC - Elliptic curve cryptography is an encryption technique using a public key. This technique is based elliptic curve theory.

3.2. Feasibility Analysis

The major deciding factor around the feasibility is that the SHARC Bouy runs on a small battery therefor to reduce battery consumption processing must be kept minimal.

As the data from the SHARC Buoy is relatively large and doesn't require complex privacy protection, we have decided on using symmetric encryption. The basis behind this decision was that the symmetric encryption was more effective on large chunks of data. Required less computational processing when compared to asymmetric encryption. On deciding which encryption algorithm would be most effective in completing the encryption, we looked at three possible encryption algorithms. Those being Two-fish, Advanced Encryption Standard and Data Encryption Standard. Upon further discussion. The decision was made that the Two-fish encryption algorithm would be used. The reason behind this decision being that the two-fish is the fastest encryption algorithm, meaning that the processing time is reduced. With the reduction in processing time. Less computational processing required, battery consumption minimal. Besides, it being the fastest encryption algorithm it also only requires one key and the key can be of lengths of up to 256 bits. This algorithm is also available to the public making it easily accessible. This is why it was concluded that the two fish algorithm would be best fit for what was required. After decompression, it is required that a minimum of 25% of the Fourier coefficients of the data be available. This makes it possible for lossless compression algorithms to be used. An advantage of using lossless compression is that the data in the file is not altered when compared to lossy compression techniques. As the main goal is getting reliable and accurate data the choice of lossless compression is preferred. The three main compression algorithms considered, are arithmetic coding, Huffman coding and Lempel-Ziv Welch. It was decided that the arithmetic coding algorithm will be used. This is mainly because it has a quicker execution time but also because it has a better complexity when compared to Huffman coding. Huffman coding has a complexity of $O(n^2)$ and arithmetic coding has a complexity of $O(n \log n)$.

3.3. Possible Bottlenecks

The phenomenon of a Bottlenecks can be defined by single component within a system limiting its performance and capacity. Within this system, the Bottlenecks can stem from components within the system. The main being the IMU, in our case the Raspberry Pi zero. The possible Bottlenecks that come from the use of the raspberry pi and the network over which the data is exchanged/ sent over are:

- The speed at which the Pi is able to process the data / algorithms.
- The bandwidth over which the data is transferred. Determines how much processing is required. Which in turn dictates the power consumption
- Inefficiencies within the algorithms used, due to the fact as they are open-source algorithm, they are not case specific and are designed for the general case.

3.4. Inter-Subsystem and Inter-Sub-Subsystem Interactions

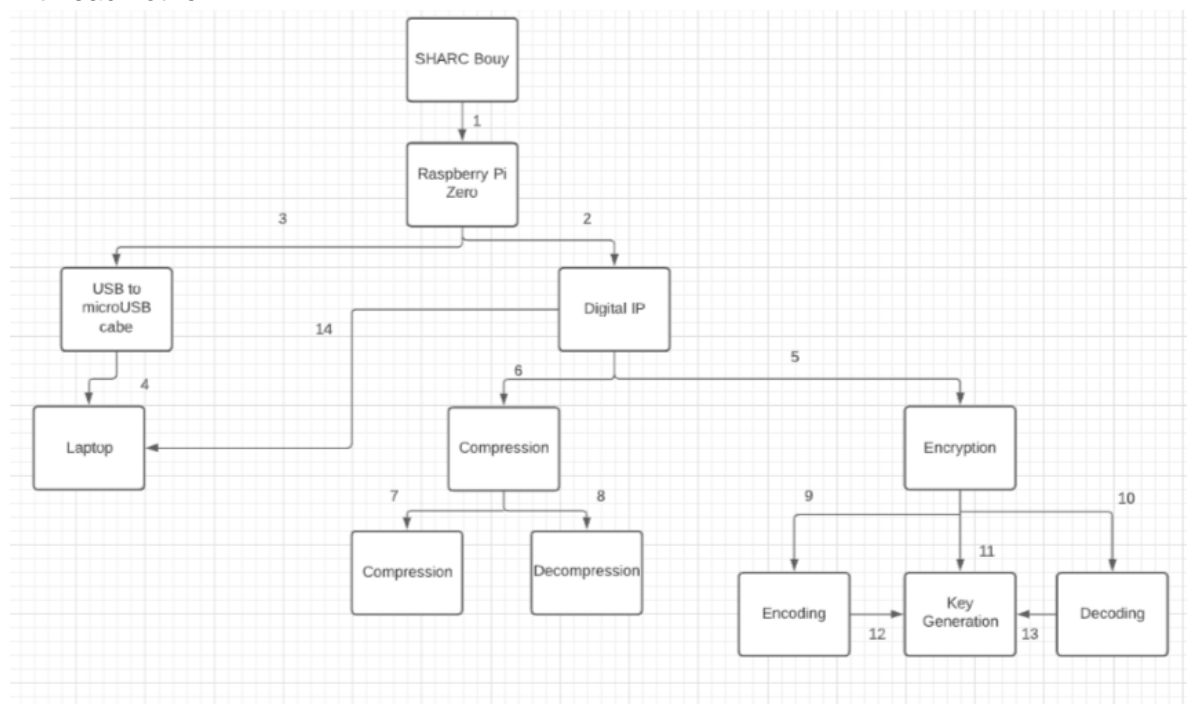
The following interactions will be explained making use of the UML diagram in question 3.4. The inter-subsystem interactions are indicated by the key "IS" and the inter-sub-subsystems by "ISS".

1. The Raspberry Pi Zero W emulates the IMU found on the SHARC Buoy. (IS)
2. The digital IP (intellectual property) runs on the raspberry pi zero w interface. (IS)
3. The Raspberry Pi Zero W is connected to the laptop via the USB cable. (IS)
4. The Laptop provides power to the raspberry pi through the USB cable. (IS)
5. The digital IP initiates encryption (IS)
6. The digital IP initiates the compression. (IS)
7. Via the arithmetic coding algorithm the data is compressed. (IS)

8. After compression occurs, the arithmetic coding algorithm decompresses the data. (IS)
9. After the Digital IP initiates encryption, data is then encrypted using encryption algorithms. (IS)
10. After the data has been sent, the encryption algorithm will perform the decryption of the data (IS)
11. The encryption algorithm will generate a key to allow for the encryption and decryption processes to occur. The encryption key is needed for any user who would like access to the data (IS)
12. The encryption algorithm will fetch the encryption key to check if the user has access to the data. If the user does the encryption algorithm is executed (ISS)
13. The encryption algorithm will fetch the encryption key to check if the user has access to the data. If the user does the encryption algorithm will then de-encrypt the data. (ISS)
14. The laptop is used to display the output after compression and encryption algorithms have executed. (ISS)

3.5. UML Diagram of System

The below diagram describes how the inter-subsystems and inter-sub-subsystems interact with each other.



4. Validation using Simulated or Old Data

4.1. Need for Simulation-Based Validation

The reason behind simulation based validation is that this provides us with a benchmark that we are then able to compare the hardware based validation to. The simulation-based validation will provide us with the most accurate results as it is directly related to the SHARC BUOY as the data is the one from the BUOY. When comparing with hardware we will be able to determine how accurately the data produced matched that of the simulations. It is a cost

effective approach when dealing with the data. As it only requires a computer and raspberry pi and no other external components and hardware

4.2. Steps

Below are the steps taken during the software-based validation:

- All the csv file provided were converted into .txt file for easier access across all platforms and easier formatting.
- The data was moved to our local git repository.
- Using our virtual machine which ran ubuntu linux. Using the “*git pull*” we were able to get all the data required.
- On the virtual machine the data was then compressed and encrypted
- The data was then pushed back to the local git repository.
- Using the raspberry pi through a laptop interface the data from the git repository was pulled.
- Now the compressed and encrypted data files were decrypted and decompressed.
- From there the changed files were then compared to the original files to see if there were any changes.
- Even though the report only shows the one data sets results. We repeated the method on different data sets of different sizes and they all displayed the same result where the files were all identical.
-

The data being used in this section is the real-world data obtained from an IMU on board a SHARC Buoy. The data is in the form of a .csv file, however it has been converted into a text file format (.txt) for ease of use when executing compression and encryption algorithms. The datasets have an average sample size of 15000, where data entries were made every 10 milliseconds.

4.3. Experimental Setup

4.3.1. System

The overall system requires that encrypted and compressed data can be sent to the processing block (in our case, the Raspberry Pi Zero) where it will be decrypted and decompressed.

The sub system is required to:

- Compress and encrypt data from a remote location
- Send the data to the processing block in real-time
- Decrypt and decompress the data on the processing block
- Data integrity after decryption and decompression must be upheld (file contents must not have changed).
- The file size of the data must have reduced in size to around 50% of the original file size
- At least 25% of the Fourier coefficients must be extracted
- The CPU and RAM usage must be kept to a minimum in order to reduce battery consumption

To test that these have been met:

- Use a virtual machine to run Ubuntu Linux this would act as a remote server.
- The raspberry pi would simulate the IMU.

- The size of the data files will be checked using the following command in terminal `"ls -lh"`.
- Then the file `data_1.txt` was compressed using the command `"gzip data_1.txt"`
- Using the command `"ls-lh"` the size of the file will be checked to see if the size of the file has actually decreased. This will indicate that the algorithm has worked. As well as to check that the file size has been reduced to less than 50%.
- After the `encryption.cpp` file has been compiled using the command `"g++ encryption.cpp -o encrypt"` the following command was run `"./encrypt"` which will prompt the user to enter the file that needs to be encrypted the file entered will be `data_1.txt.gz` this will then encrypt the file.
- Then the contents of the file will be checked using `"sudo joe data_1.txt.gz"` this will show if the data within the file has been encrypted.
- From there the data will be added, committed and pushed to the git repository.
- The raspberry pi will be setup and it will pull from the git repository. Using the git pull command.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` This is used to check that the CPU usage is below 50%
- This will take note of what the average CPU usage is when the pi is idle
- Now that the data file is on the raspberry pi. The `decryption.cpp` will be executed similarly to the `encryption.cpp` using the command `"g++ decryption.cpp -o decrypt"` after this has been executed and compiled properly
- The command `"./decrypt"` will be run this will then prompt the user to enter the file they want to decrypt. To which the file `"data_1.txt.gz"` will be entered after the decryption is complete.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` this was done to check how the average CPU usage changed after this command was run. This is used to check that the CPU usage is below 50%
- Next the command `"ls -lh"` was run to check that the size of the file had not change since the decryption algorithm was executed.
- Next the command `"gunzip data_1.txt.gz"` was run this will then decompress the file thus returning the original file.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` was run to check how the CPU usage was affected after the decompression was done. This is used to check that the CPU usage is below 50%
- The command `"ls-lh"` was executed to check the size of the file and compare it to the size of the original data file. This will be an indicator to show that all processes have worked and the original file can be retrieved from all these processes.
- Lastly this file will be taken onto a windows operation based laptop to compare the integrity of the files. Using the command `"if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}"` in windows powershell. This command will return if the files are identical or not.
- If the files are identical this will mean that the fourier plot will be the same. So the fourier coefficients will be the same value. If they are not identical the data will be sent to matlab and the coefficient checked and as to what degree the coefficient is decreased to.

4.3.2. Compression [CRTBRA002]

The compression subsystem requires that data be compressed and decompressed.

The sub system is required to:

- Compress the data
- Decompress the data
- Data integrity after decompression must be upheld (file contents must not have changed).
- The file size of the data must have reduced in size to around 50% of the original file size
- The CPU and RAM usage must be kept to a minimum in order to reduce battery consumption
- At least 25% of the Fourier coefficients must be extracted

To test that these have been met:

- The size of the data files will be checked using the following command in terminal `"ls -lh"`. Check the size of the file.
- Then the file `data_1.txt` was compressed using the command `"gzip data_1.txt"`
- Using the command `"ls-lh"` the size of the file will be checked to see if the size of the file has actually decreased. This will indicate that the algorithm has worked. As well as to check that the file size has been reduced to less than 50%.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` This is used to check that the CPU usage is below 50%
- Next the command `"gunzip data_1.txt.gz"` was run this will then decompress the file thus returning the original file.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` was run to check how the CPU usage was affected after the decompression was done. This is used to check that the CPU usage is below 50%
- The command `"ls-lh"` was executed to check the size of the file and compare it to the size of the original data file. To check if the data has been restored to the original size.
- Lastly this file will be taken onto a windows operation based laptop to compare the integrity of the files. Using the command `"if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}"` in windows powershell. This command will return if the files are identical or not.
- If the files are identical this will mean that the fourier plot will be the same. So the fourier coefficients will be the same value. If they are not identical the data will be sent to matlab and the coefficient checked and seen to what degree the coefficient is decreased to.

4.3.3. Encryption [THMNAT011]

The encryption subsystem requires that data be encrypted and decrypted.

The sub system is required to:

- Encrypt the data
- Decrypt the data
- Data integrity after decryption must be upheld (file contents must not have changed).
- The CPU and RAM usage must be kept to a minimum in order to reduce battery consumption

To test that these have been met:

- The size of the data files will be checked using the following command in terminal `"ls -lh"`. Check the size of the file.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` this is to be used to check the CPU usage at the idle state.
- After the encryption.cpp file has been compiled using the command `"g++ encryption.cpp -o encrypt"` the following command was run `"./encrypt"` which will prompt the user to enter the file that needs to be encrypted the file entered will be `data_1.txt.gz` this will then encrypt the file.
- Then the contents of the file will be checked using `"sudo nano data_1.txt"` this will show if the data within the file has been encrypted.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` This is used to check that the CPU usage is below 50%
- Now that the data file is on the raspberry pi. The decryption.cpp will be executed similarly to the encryption.cpp using the command `"g++ decryption.cpp -o decrypt"` after this has been executed and compiled properly
- The command `"./decrypt"` will be run this will then prompt the user to enter the file they want to decrypt. To which the file `"data_1.txt"` will be entered after the decryption is complete.
- Next the avg CPU usage will be checked `"sudo iostat -c 2 2"` This is used to check that the CPU usage is below 50%
- Then the contents of the file will be checked using `"sudo nano data_1.txt"` this will show if the data within the file has been decrypted.
- Lastly this file will be taken onto a windows operation based laptop to compare the integrity of the files. Using the command `"if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}"` in windows powershell. This command will return if the files are identical or not.

4.4. Results

4.4.1. System

These are the results from the remote computer (running Ubuntu Linux) during file compression and encryption:

- ls -lh #Check file size before compression

```

$ ls -lh
total 18M
-rw-rw-r-- 1 nathanael nathanael 8.7M Oct 4 22:10 data_1.txt
-rwxrwxr-x 1 nathanael nathanael 19K Oct 4 22:06 decrypt
-rwxrwxr-x 1 nathanael nathanael 728 Oct 4 20:35 decryption.cpp
-rwxrwxr-x 1 nathanael nathanael 18K Oct 4 22:06 encrypt
-rwxrwxr-x 1 nathanael nathanael 19K Oct 4 20:44 encrypt.exe
-rw-rw-r-- 1 nathanael nathanael 1.1K Oct 4 20:35 encryption.cpp
-rw-rw-r-- 1 nathanael nathanael 8.7M Oct 4 22:10 tmp.txt

```

- gzip data_1.txt #Compression
- ls -lh #Check file size after compression

```
er$ ls -lh
total 13M
-rw-rw-r-- 1 nathanael nathanael 3.7M Oct 4 22:10 data_1.txt.gz
-rwxrwxr-x 1 nathanael nathanael 19K Oct 4 22:06 decrypt
-rw-rw-r-- 1 nathanael nathanael 728 Oct 4 20:35 decryption.cpp
-rwxrwxr-x 1 nathanael nathanael 14K Oct 4 22:10 encrypt
-rw-rw-r-- 1 nathanael nathanael 19K Oct 4 20:44 encrypt_exe
-rw-rw-r-- 1 nathanael nathanael 1.1K Oct 4 20:35 encryption.cpp
-rw-rw-r-- 1 nathanael nathanael 8.7M Oct 4 22:19 tno_1st
```

- ```
- ./encrypt #Encryption
- joe data_1.txt.gz #Check the contents of the file after encryption
```

[illegible]

Then from there the data was added and committed to our public GitHub repository where it will be pulled by the Raspberry Pi Zero W.

These are the results that were collected from the Raspberry Pi Zero W after decryption and decompression:

- `sudo iostat -c 2 2` #Check the average CPU before decryption and decompression



```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: user nice system iowait steal idle %iowait
 5.31 0.00 1.10 0.15 0.00 93.44

avg-cpu: user nice system iowait steal idle %iowait
 0.00 0.00 1.00 0.00 0.00 99.00

```

- ./decrypt #Decryption
- sudo iostat -c 2 2 #Check the average CPU usage after decryption

```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: user nice system iowait steal idle %iowait
 5.40 0.00 1.10 0.15 0.00 93.35

avg-cpu: user nice system iowait steal idle %iowait
 0.50 0.00 1.01 0.00 0.00 99.50

```

- gunzip data\_1.txt.gz #Decompression
- sudo iostat -c 2 2 #Check the average CPU usage after decompression

```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: user nice system iowait steal idle %iowait
 5.37 0.00 1.10 0.15 0.00 93.38

avg-cpu: user nice system iowait steal idle %iowait
 0.00 0.00 1.01 0.00 0.00 99.99

```

- ls -lh #Check the size of the file after decompression

```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ ls -lh
total 12M
-rw-r--r-- 1 pi pi 8.7M Oct 4 21:23 data_1.txt
-rwxr-xr-x 1 pi pi 14K Oct 4 21:23 decrypt
-rw-r--r-- 1 pi pi 728 Oct 4 21:25 decryption.cpp
-rwxr-xr-x 1 pi pi 19K Oct 4 21:21 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encryption.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 3.7M Oct 4 21:21 tmp.txt

```

- joe data\_1.txt #Check the contents of the file

```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker
GNU nano 3.2
data_1.txt

computer utc start 2018-09-19-03:57:21.506044
Utc computer time, Mhz, Mhz, Mhz, Acc, Acc, Gyro, Gyro, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM2, DCM3, DCM4, DCM5, DCM6, DCM7, DCM8, DCM9, MagNED1, MagNED2, MagNED3, Az
2018-09-19-03:57:21.717926 -0.14322271943092346 0.2232052981853485 0.123428575694561 0.15337686579227448 -0.30159556216812134 -0.795899391174316 0.0508350133895874 -0.0013912274735048413 5
2018-09-19-03:57:21.817726 -0.1410953253507614 0.21974442899227142 0.12359068542718887 0.15984362363815308 -0.30425748229026794 -0.779479026794434 0.050224707908742189 -0.00184811000712215
2018-09-19-03:57:21.917707 -0.1453813448425108 0.224339616518563843 0.12579116225242615 0.159090339942398 -0.2982486046684265 -0.78246116381836 0.00462750381082296 -0.000948803790379315
2018-09-19-03:57:22.017726 -0.14480726232359922 0.2198091596344975 0.1245958544008416 0.15925079584121704 -0.3047791719436455 -0.774477005004883 0.00507897929294553 -0.002422330688845
2018-09-19-03:57:22.117717 -0.14319761097431183 0.2232208251953125 0.1271720826425824 0.1546937552723694 -0.38872249603271484 -0.755866050720215 0.00510706853693247 -0.000864562889561055
2018-09-19-03:57:22.217686 -0.14326296746738084 0.2209255630455017 0.12470496445894241 0.1608011265182495 -0.30884605646133423 -0.749424934387207 0.004865021910518408 -0.00116990955933695
2018-09-19-03:57:22.317733 -0.14428161084651947 0.22437554597854614 0.12585045397281647 0.15797586739063263 -0.3115788400173187 -0.74105453481211 0.004976013209670782 -0.001297013368457555
2018-09-19-03:57:22.417797 -0.146558244457245 0.220964750234024 0.12574532556338 0.1628453450414605 -0.31055364753723 -0.738278763343961 0.00463604317074722 -0.00125477093641124 5
2018-09-19-03:57:22.517726 -0.1454533731378885 0.220974706244348 0.1258334664490548 0.16180342605113983 -0.3162236213684082 -0.72849089467773 0.004644147580077452 -0.00177446355144078
2018-09-19-03:57:22.617713 -0.1487094089552713 0.22331589460372925 0.125627070663595 0.16106994450092316 -0.31497976183891296 -0.719582557678223 0.004643251188099384 -0.000851867851077165
2018-09-19-03:57:22.717685 -0.14646583795547485 0.22555047273635864 0.12446887791156769 0.16358499228954315 -0.3218514621257782 -0.71753215789795 0.00438275059573689 -0.000835037811473015
2018-09-19-03:57:22.817678 -0.1466141045035364 0.21983402967453003 0.12079749256372452 0.15981677174568176 -0.32272082567214966 -0.700324058532715 0.004025812260805605 -0.0013239498484678
2018-09-19-03:57:22.917676 -0.14653174579143524 0.22232601195573068 0.12200223124846949 0.1651551372491365 -0.3259122667720232 -0.702162284719668 0.0040995571722238 -0.0012727394051929
2018-09-19-03:57:23.017675 -0.1465712934732437 0.22098377346992493 0.1232781112194013 0.16342021524906158 -0.326689675807953 -0.694110870361328 0.003974369261413813 -0.001242150075763465
2018-09-19-03:57:23.117681 -0.145454287528917 0.22209881246089935 0.12207549065351486 0.166680080172989 -0.31716904044151306 -0.68447494506836 0.003931220155299303 -0.00124326010700315245
2018-09-19-03:57:23.217676 -0.1496333960715607 0.2233194771957397 0.1218237653374672 0.1771417072048187 -0.323736983394623 -0.685702323913574 0.00367445987049103 -0.00120609958380805
2018-09-19-03:57:23.317669 -0.147861370162996 0.221015223324081 0.1218117844663925 0.1770520808312561 -0.323736983394623 -0.685702323913574 0.00367445987049103 -0.00120609958380805
2018-09-19-03:57:23.417658 -0.1465200316963837 0.22213870286941528 0.12700651586055756 0.1742392378369604 -0.32891881465911865 -0.6724962993164 0.0033595361659348 -0.0015583423664793375
2018-09-19-03:57:23.517672 -0.14755785701006775 0.22103434801101685 0.12565478682518005 0.17514801025390625 -0.3213691711425781 -0.66642802886629 0.002892849745030157 -0.00168299640061395
2018-09-19-03:57:23.617666 -0.14763095480187256 0.2221539765596398 0.1256995201108396 0.17808954417705536 -0.319258397960663 -0.660322189331055 0.0025873205025022462 -0.001564605743624275
2018-09-19-03:57:23.717660 -0.147641524242426 0.2246963284172974 0.1260593830041885 0.1737076153094235 -0.324848998065763 -0.6681747435234 0.002747015794739127 -0.001415552431972025
2018-09-19-03:57:23.817687 -0.147620120584488 0.22158282895224 0.1269466367099762 0.1758901327848434 -0.31988441944123314 -0.65793323168457 0.00282432864785194 -0.0017178648300468925
2018-09-19-03:57:23.917668 -0.14863012731075287 0.22673363963631134 0.12558147311210632 0.17027561366580755 -0.320859090576172 -0.656055450439453 0.00290258289991187 -0.00181777032225685
2018-09-19-03:57:24.017660 -0.1486874678134918 0.22332346439361572 0.12812024354934692 0.1735814840316772 -0.3253801465034408 -0.66348648071289 0.0020196966361254454 -0.00188359862328635
2018-09-19-03:57:24.117657 -0.1519823525926283 0.22451435029506683 0.1254326403140217 0.17471052706241608 -0.325307640552521 -0.666720390319624 0.001436846261862888 -0.001964975190192415
2018-09-19-03:57:24.217683 -0.14975623718648432 0.22561382188072205 0.1255349184013672 0.17431404545292914 -0.3253460824489595 -0.66325374179488 0.00124470400220019 -0.002144448481325
2018-09-19-03:57:24.317674 -0.14965781569480896 0.22905002534389496 0.1282371153831482 0.17736057937145233 -0.32447412610054016 -0.658225055905277 0.001014639623615158 -0.00216436828466622
2018-09-19-03:57:24.417646 -0.1507580280303555 0.2296970977783203 0.1291773022122161 0.1757822483778 -0.3206561505794525 -0.659080979797363 0.0009066941565833986 -0.001995852665845856 -05
2018-09-19-03:57:24.517657 -0.1497454441547394 0.225639005350113 0.13177543878555296 0.1710270156639497 -0.316857286643982 -0.66127863647461 0.00087008671660651 -0.002415154457937178
2018-09-19-03:57:24.617692 -0.14984577984210815 0.221059514503479 0.12684187293052670 0.17468910450172424 -0.3130212128122304 -0.673789978027344 0.000570644857361927 -0.0025147900450984782
2018-09-19-03:57:24.717629 -0.148612307244873 0.2256136685698175 0.1305865198373946 0.1742662939548492 -0.31707286834716797 -0.675750732421675 0.0003847305489307029 -0.002355016713979285
2018-09-19-03:57:24.817658 -0.149050247736609 0.22220884263515472 0.12923243780994415 0.17096800156116486 -0.31041204929351807 -0.66987895955762 0.000545960199006 -0.0022314775264785
2018-09-19-03:57:24.917692 -0.14973071217336928 0.224503993980371 0.13303738832479755 0.17257314920425415 -0.3047507243156433 -0.6892242411640627 0.0008374145582856 -0.00231140037067235
2018-09-19-03:57:25.017660 -0.1497454441547394 0.225639005350113 0.13177543878555296 0.1710270156639497 -0.316857286643982 -0.66127863647461 0.00087008671660651 -0.002415154457937178
2018-09-19-03:57:25.117687 -0.14984577984210815 0.221059514503479 0.12684187293052670 0.17468910450172424 -0.3130212128122304 -0.673789978027344 0.000570644857361927 -0.0025147900450984782
2018-09-19-03:57:25.217646 -0.1518764299969537 0.22905002534389496 0.1282371153831482 0.17736057937145233 -0.32447412610054016 -0.658225055905277 0.001014639623615158 -0.00216436828466622
2018-09-19-03:57:25.317639 -0.1476193517446518 0.22103284299373627 0.130703911852466 0.17632925211055756 -0.2950594909145405 -0.70406436920166 -0.0007328435313956372 -0.002673068549484015
2018-09-19-03:57:25.417659 -0.1530457358971746 0.224544748140385 0.1308267772795392 0.1707105006596963 -0.290140593515167 -0.72524642444336 -0.00026353220972581 -0.00281700796544545
2018-09-19-03:57:25.517643 -0.15086373686790466 0.22450338304042816 0.1278665463924408 0.16665786504774583 -0.29252463579177856 -0.732711791962188 -0.0009374540532007813 -0.002890572650355
2018-09-19-03:57:25.617672 -0.14977217467633057 0.22563363169815 0.130527526140213 0.1697662841777802 -0.2871216833591461 -0.71875476371582 -0.00039953542923123697 -0.003074936801567675
2018-09-19-03:57:25.717628 -0.14968096945705414 0.21875249811172485 0.13061365485193345 0.16782568395137787 -0.2934019863605499 -0.740289688110352 -0.0007951246807531708 -0.00253118271393945
2018-09-19-03:57:25.817627 -0.150452459049225 0.22110219299753243 0.130202413963918 0.1664672919578552 -0.2907872200012207 -0.747501481933594 -0.000672321008953221 -0.0028792232063319
2018-09-19-03:57:25.917610 -0.14989322423394937 0.2187771440165639 0.1268699765203833 0.168485937468444 -0.284744335219574 -0.754037557055664 -0.000697533997744322 -0.0028701316164608
2018-09-19-03:57:26.017622 -0.149789348859787 0.2187727987766266 0.13067327439875004 0.165043857550656 -0.28077906370162964 -0.761462211608887 -0.0005838487283906758 -0.002773119053999755
2018-09-19-03:57:26.117612 -0.14978736639028827 0.22221824526786804 0.1318181080100506 0.168884024024097 -0.2796405959129333 -0.76777535583496 -0.0006195941823534667 -0.0028288112953305

```

- Compare the contents of the original and extracted files

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Adarsh> if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}

Both the files are identical
PS C:\Users\Adarsh>

```

#### 4.4.2. Compression [CRTBRA002]

These are the results of the independent testing of the compression algorithm on the Raspberry Pi Zero W:

- `ls -lh` #Check the file size before compression

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ ls -lh
total 13M
-rw-r--r-- 1 pi pi 8.7M Oct 4 20:39 data_1.txt
-rwxr-xr-x 1 pi pi 14K Oct 3 16:56 decrypt
-rw-r--r-- 1 pi pi 728 Oct 3 16:55 decryption.cpp
-rwxr-xr-x 1 pi pi 14K Oct 4 20:17 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encrypt.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 3.7M Oct 4 20:30 tmp.txt
```

- `sudo iostat -c 2 2` #Check the average CPU usage before compression

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 3.23 0.00 1.24 0.21 0.00 95.32

avg-cpu: %user %nice %system %iowait %steal %idle
 0.50 0.00 1.00 0.00 0.00 98.51
```

- `gzip data_1.txt` #Compression

- `sudo iostat -c 2 2` #Check the average CPU usage after compression

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 3.50 0.00 1.23 0.21 0.00 95.06

avg-cpu: %user %nice %system %iowait %steal %idle
 0.50 0.00 0.50 0.00 0.00 98.99
```

- `ls -lh` # Check the file size after compression

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ ls -lh
total 7.3M
-rw-r--r-- 1 pi pi 3.7M Oct 4 20:39 data_1.txt.gz
-rwxr-xr-x 1 pi pi 14K Oct 3 16:56 decrypt
-rw-r--r-- 1 pi pi 728 Oct 3 16:55 decryption.cpp
-rwxr-xr-x 1 pi pi 14K Oct 4 20:17 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encrypt.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 3.7M Oct 4 20:30 tmp.txt
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $
```

- `nano data_1.txt.gz` #Check the file contents after compression



```

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker
GNU nano 3.2 data_1.txt.gz
... (C code for decompression) ...
}

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ gunzip data_1.txt.gz #Decompression
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 3.37 0.00 1.19 0.20 0.00 95.24

avg-cpu: %user %nice %system %iowait %steal %idle
 0.00 0.00 0.50 0.00 0.00 99.50

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ ls -lh
total 13M
-rw-r--r-- 1 pi pi 8.7M Oct 4 20:39 data_1.txt
-rwxr-xr-x 1 pi pi 14K Oct 3 16:56 decrypt
-rw-r--r-- 1 pi pi 728 Oct 3 16:55 decryption.cpp
-rwxr-xr-x 1 pi pi 14K Oct 4 20:17 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encrypt.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 3.7M Oct 4 20:30 tmp.txt

pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $

```

- gunzip data\_1.txt.gz #Decompression

- sudo iostat -c 2 2 #Check the average CPU usage after decompression

- ls -lh # Check the file size after decompression

- Compare the contents of the original and extracted files

```

PS C:\Users\Adarsh> if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}

Both the files are identical
PS C:\Users\Adarsh>

```

#### 4.4.3. Encryption [THMNAT011]

These are the results of the independent testing of the encryption algorithm on the Raspberry Pi Zero W:

- `ls -lh` #Check the file size before encryption

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ ls -lh
total 13M
-rw-r--r-- 1 pi pi 8.7M Oct 4 20:39 data_1.txt
-rwxr-xr-x 1 pi pi 14K Oct 3 16:56 decrypt
-rw-r--r-- 1 pi pi 728 Oct 3 16:55 decryption.cpp
-rwxr-xr-x 1 pi pi 14K Oct 4 20:17 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encrypt.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 3.7M Oct 4 20:30 tmp.txt
```

- `sudo iostat -c 2 2` #Check the average CPU usage before encryption

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 3.27 0.00 1.17 0.19 0.00 95.37

avg-cpu: %user %nice %system %iowait %steal %idle
 3.00 0.00 1.00 0.00 0.00 96.00
```

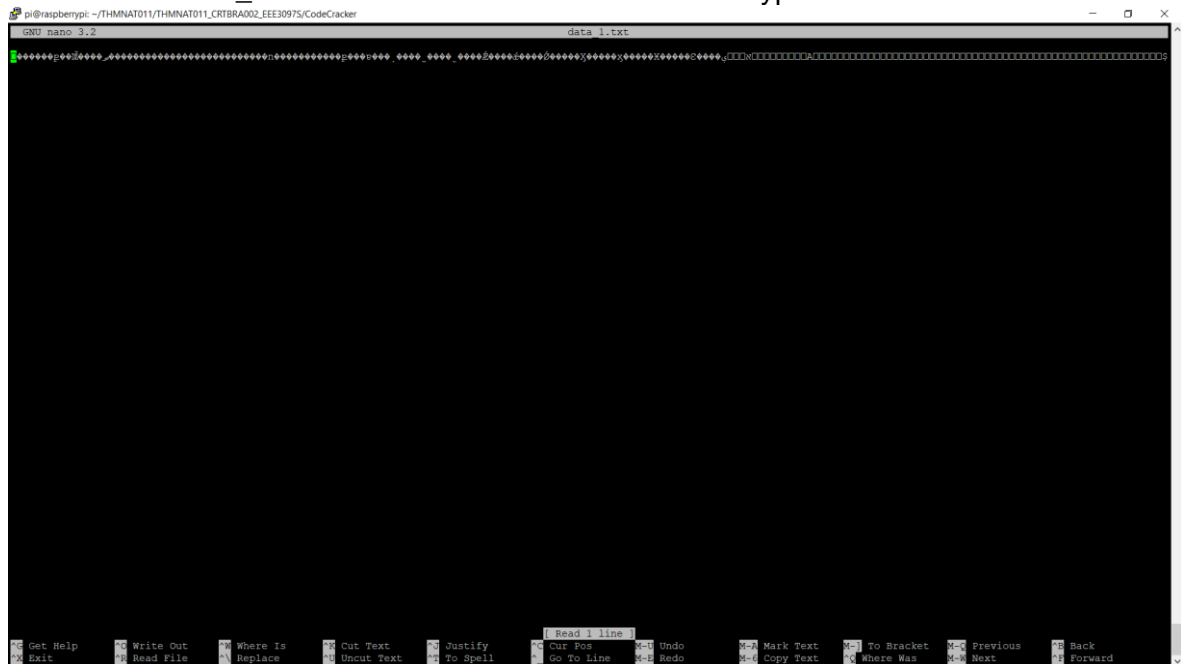
- `./encrypt` #Encryption
- `sudo iostat -c 2 2` #Check the average CPU usage after encryption

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 3.52 0.00 1.18 0.19 0.00 95.11

avg-cpu: %user %nice %system %iowait %steal %idle
 3.00 0.00 0.81 0.00 0.00 96.19
```

- `nano data_1.txt` #Check the file contents after encryption



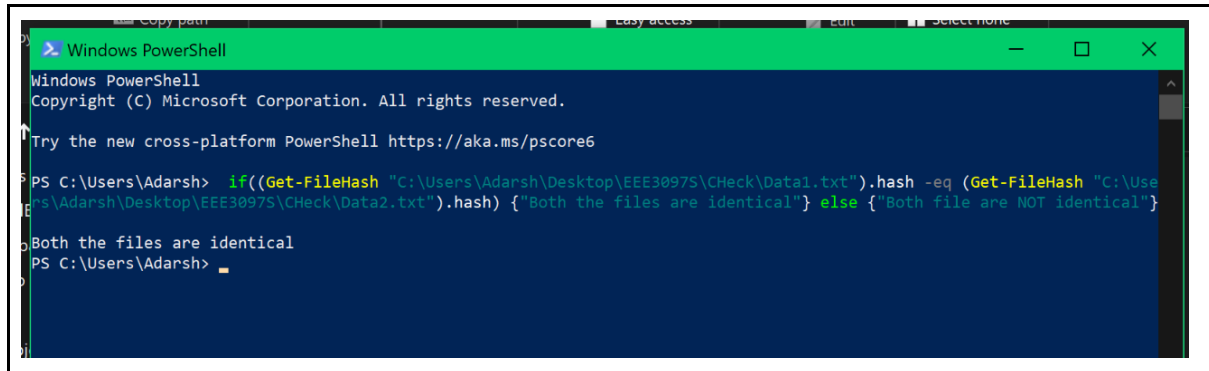
- `gunzip data_1.txt.gz` #Decryption
- `sudo iostat -c 2 2` #Check the average CPU usage after decryption

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker $ sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 04/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 4.03 0.00 1.16 0.19 0.00 94.63

avg-cpu: %user %nice %system %iowait %steal %idle
 1.01 0.00 0.81 0.00 0.00 98.18
```

- Compare the contents of the original and extracted files



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Adarsh> if((Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data1.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\EEE3097S\Check\Data2.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}

Both the files are identical
PS C:\Users\Adarsh>
```

## 5. Validation using a different-IMU

### 5.1. Need for Hardware-Based Validation

The reason behind hardware based validation is to check that the software based system can be implemented in the real-world and is not just a valid solution from a software perspective. It is also vital that we are able to generate our own data from the surrounding environment in real-time and not just use data provided to us, as the SHARC Buoy is required to generate data from the wave movement and pancake ice in real-time too. The software design also needs to be feasible and cost-effective in reality, this is another reason why a hardware solution must be implemented.

### 5.2. Steps

Below are the steps taken during the hardware-based validation:

- Generated data using the attached Sense-HAT B
- Compress and encrypt the data on the Raspberry Pi Zero W
- Send the compressed and encrypted data across to a remote computer using GitHub
- Decrypt and decompress the data on the remote laptop
- The original file and extracted file are compared to check for data loss
- Average CPU usage is monitored throughout the execution of the algorithms
- Files of varying sizes were used to test the algorithms (only one case was shown in the results section to save space and avoid repetition).

The data being used in this section is the data generated from the Sense-HAT B attached to the Raspberry Pi Zero. The data is extracted in the form of a text file format (.txt) for ease of use when executing compression and encryption algorithms. The datasets can have a sample size of any size, this is determined by the user. However the larger the dataset, the longer it will take to generate the file. File generation has an average rate of 2.5 samples per second.

### 5.3. Experimental Setup

#### 5.3.1. System

The overall system requires that data from a an IMU sensor be encrypted and compressed on a processing block (in our case, the Raspberry Pi Zero) and sent to a remote location where it can be decrypted and decompressed.

The sub system is required to:

- Compress and encrypt data from on the processing block
- Send the data to a remote location in real-time



- Decrypt and decompress the data on the remote computer
- Data integrity after decryption and decompression must be upheld (file contents must not have changed).
- The file size of the data must have reduced in size to around 50% of the original file size
- At least 25% of the Fourier coefficients must be extracted
- The CPU and RAM usage must be kept to a minimum in order to reduce battery consumption

To test that these have been met:

- Use “`sudo python ICM20948.py`” to run the program that generates data from the sensors on the IMU. After this command is run the user must enter the sample size in the next line for the program to begin (for example, “15000”). To append this data to a file, use the command “`sudo python ICM20948.py > 15k_data.txt`”.
- Use a virtual machine to run Ubuntu Linux this would act as a remote server.
- The raspberry pi would simulate the IMU.
- The size of the data files will be checked using the following command in terminal “`ls -lh`”.
- Then the file `data_1.txt` was compressed using the command “`gzip 15k_data.txt`”
- Using the command “`ls -lh`” the size of the file will be checked to see if the size of the file has actually decreased. This will indicate that the algorithm has worked. As well as to check that the file size has been reduced to less than 50%.
- After the `encryption.cpp` file has been compiled using the command “`g++ encryption.cpp -o encrypt`” the following command was run “`./encrypt`” which will prompt the user to enter the file that needs to be encrypted the file entered will be `data_1.txt.gz` this will then encrypt the file.
- Then the contents of the file will be checked using “`sudo joe 15k_data.txt.gz`” this will show if the data within the file has been encrypted.
- From there the data will be added, committed and pushed to the git repository.
- The raspberry pi will be setup and it will pull from the git repository. Using the git pull command.
- Next the avg CPU usage will be checked “`sudo iostat -c 2 2`” This is used to check that the CPU usage is below 50%
- This will take note of what the average CPU usage is when the pi is idle
- Now that the data file is on the raspberry pi. The `decryption.cpp` will be executed similarly to the `encryption.cpp` using the command “`g++ decryption.cpp -o decrypt`” after this has been executed and compiled properly
- The command “`./decrypt`” will be run this will then prompt the user to enter the file they want to decrypt. To which the file “`15k_data.txt.gz`” will be entered after the decryption is complete.
- Next the avg CPU usage will be checked “`sudo iostat -c 2 2`” this was done to check how the average CPU usage changed after this command was run. This is used to check that the CPU usage is below 50%
- Next the command “`ls -lh`” was run to check that the size of the file had not changed since the decryption algorithm was executed.
- Next the command “`gunzip 15k_data.txt.gz`” was run this will then decompress the file thus returning the original file.

- Next the avg CPU usage will be checked “*sudo iostat -c 2 2*” was run to check how the CPU usage was affected after the decompression was done. This is used to check that the CPU usage is below 50%
- The command “*ls-lh*” was executed to check the size of the file and compare it to the size of the original data file. This will be an indicator to show that all process have worked and the original file can be retrieved from all these processes.
- Lastly this file will be taken onto a windows operation based laptop to compare the integrity of the files. Using the command “*if((Get-FileHash "C:\Users\Adarsh\Desktop\Git\THMNAT011\_CRTBRA002\_EEE3097\check\15k\_data.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\Git\THMNAT011\_CRTBRA002\_EEE3097\check\data1.txt").hash) {"Both the files are identical"} else {"Both file are NOT identical"}"* in windows powershell. This command will return if the files are identical or not.
- If the files are identical this will mean that the fourier plot will be the same. So the fourier coefficients will be the same value. If they are not identical the data will be sent to matlab and the coefficient checked and a to what degree the coefficient is decreased to.

**Comparison of the sub-systems individually the tests were run purely on the raspberry pi.**

### 5.3.2. Compression [CRTBRA002]

To avoid repetition in the experimental setup, this section will be left out, therefore refer to Section 4.3.2 for the independent setup of the compression algorithm. This conclusion was made because there is little to no variation in the independent setup of the compression algorithm.

### 5.3.3. Encryption [THMNAT011]

To avoid repetition in the experimental setup, this section will be left out, therefore refer to Section 4.3.3 for the independent setup of the encryption algorithm. This conclusion was made because there is little to no variation in the independent setup of the encryption algorithm.

## 5.4. Results

### 5.4.1. System

These are the results from the Raspberry Pi Zero during file generation, compression and encryption.

- *sudo python ICM20948.py* #Show the generation of sensor data

```
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/Sense-HAT-B-Demo/ICM-20948/Raspberry Pi/python $ sudo python ICM20948.py
10
roll pitch yaw AccX AccY AccZ GyroX GyroY GyroZ MagX MagY MagZ
-0.23653140381 0.0713921127517 -175.034036486 26 226 16749 3 1 1 86 -31 9
-0.278144492283 0.109785235653 -172.803472484 1 225 16802 1 0 -1 87 -29 8
-0.0208251948648 0.0816184568937 -172.804952277 5 218 16807 2 0 -1 85 -31 8202
-0.15453194998 0.126163153838 -170.00201037 25 209 16820 1 1 0 86 -32 10
-0.147742247247 0.142089996985 -168.470540203 9 225 16815 1 2 -1 87 -27 12
-0.18754593133 0.183948042558 -166.144824014 7 211 16819 4 3 0 87 -36 10
-0.0862248676739 0.173088307524 -165.047489443 18 268 16832 2 2 1 84 -29 12
-0.0322383501679 0.155890882138 -164.014522362 23 220 16827 0 -1 1 86 -31 12
0.0616860814974 0.121793087471 -163.194966171 37 229 16831 1 1 1 86 -31 10
0.178210019572 0.102908386255 -162.889700813 8 224 16830 3 1 1 88 -29 13
pi@raspberrypi:~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/Sense-HAT-B-Demo/ICM-20948/Raspberry Pi/python $
```

- *ls -lh* #Check file size before compression

```

-rw-r--r-- 1 pi pi 1.2M Oct 23 17:56 15k_data.txt
-rw-r--r-- 1 pi pi 1.8K Oct 22 14:43 cpdata.txt
-rw-r--r-- 1 pi pi 8.7M Oct 4 21:23 data.1.txt
-rw-r--r-- 1 pi pi 994 Oct 22 14:46 data.15k.gz
-rwxr-xr-x 1 pi pi 14K Oct 4 21:23 decrypt
-rw-r--r-- 1 pi pi 728 Oct 3 16:55 decryption.cpp
-rwxr-xr-x 1 pi pi 14K Oct 22 14:46 encrypt
-rwxr-xr-x 1 pi pi 19K Oct 4 20:03 encryption.exe
-rw-r--r-- 1 pi pi 1.1K Oct 3 16:49 encryption.cpp
-rw-r--r-- 1 pi pi 994 Oct 22 14:46 tmp.txt

```

- nano 15k\_data.txt #Check the contents of the file before compression

```

pi@raspberrypi: ~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/Sense-HAT-8-Demo/ICM-20948/Raspberry Pi/python
GNU nano 3.2 15k_data.txt
011 pitch yaw AccX AccY AccZ GyrX GyrY GyrZ MagX MagY MagZ
2.33576326339 -2.43669036112 -173.84670029 -4948 2995 16960 -3722 -834 -657 23 -79 -67
-4.82101619337 -0.266946702966 -162.338445303 -167 -8556 9394 533 453 -684 30 -16422 -33
8.17903893837 3.46779415274 -174.561935343 -4981 13307 20148 4628 -341 -204 54 16399 -6
3.82401237958 -2.25014983142 -169.70288043 -3485 6976 17636 -7158 -4094 360 65 -65 -54
0.44992906328 -6.36164910701 -166.885873557 2373 -2343 12202 -1243 -1257 -82 97 -59 -67
9.2347396934 -22.3440195985 175.211679857 13316 3068 11114 2458 -6557 -2295 155 8206 -180
26.2496697092 -37.0416458288 156.174704768 9432 11348 -16592 2130 -8083 -6195 77 34 -284
42.3023023955 -32.2852463179 133.949084227 -1045 5705 -13665 572 -1409 -2347 45 16390 -284
69.1886323183 -45.5607569119 109.64843743 -5728 300 -14329 3921 -6804 306 -8230 -15 -241
91.7092846989 -39.7061891209 97.6311650935 -17715 244 -4496 1549 -6547 2393 -76 -16388 -148
81.5833261869 -25.1475959546 95.7406980077 -15356 2323 6544 113 -2660 1079 -70 16396 -180
69.6467179003 -14.3037562052 117.639690538 -11901 6249 -7187 -7252 13897 2998 96 92 -232
68.1223080891 -25.8697621966 138.84671776 3314 9164 -5614 -8261 10848 11615 119 -8243 -87
53.6208694073 -29.7956810338 140.384408554 4467 -7393 17743 -985 -1448 1940 90 -83 -85
41.7115943148 -35.5729263704 155.624467814 8735 -8693 19414 -996 -3348 1447 75 -96 -97
37.2678272334 -47.3097158918 160.764991712 6904 -2588 1216 -1137 -2192 225 75 -96 -101
25.0352934136 -42.6676723127 179.442501406 7665 -6528 15612 1181 2785 1576 43 -105 -87
10.1541724808 -50.6176639528 -165.609197486 4670 -6498 16628 1581 366 2305 -16391 -114 -51
-7.69201175664 -52.6606728343 -150.235449867 -3444 -8544 13338 -536 -421 736 -8201 -117 -52
-6.38703126905 -42.0380964129 -150.766508246 898 -6857 14116 -162 1457 -3950 43 -76 -65
-3.95151337876 -34.8721685323 -148.917514928 3372 -227 17468 434 214 -702 46 -50 -25
-1.73932366168 -32.4478410814 -147.230190335 6315 2477 18609 -45 -659 493 52 -53 -37
-3.86104222765 -27.8005046235 -146.911045972 423 -2118 15310 -549 -702 3625 24577 -103 -54

```

- sudo iostat -c 2 2 #Check the CPU usage before compression

```

pi@raspberrypi: ~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker 8 sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 23/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 1.99 0.00 1.26 0.13 0.00 96.63

avg-cpu: %user %nice %system %iowait %steal %idle
 0.00 0.00 1.00 0.00 0.00 99.00

```

- gzip 15k\_data.txt #Compression

- sudo iostat -c 2 2 #Check CPU usage after compression

```

pi@raspberrypi: ~/THMNAT011/THMNAT011_CRTBRA002_EEE3097S/CodeCracker 8 sudo iostat -c 2 2
Linux 5.10.17+ (raspberrypi) 23/10/21 _armv6l_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 2.04 0.00 1.26 0.13 0.00 96.58

avg-cpu: %user %nice %system %iowait %steal %idle
 0.00 0.00 1.01 0.00 0.00 99.49

```

- nano 15k\_data.txt.gz #Check file contents after compression



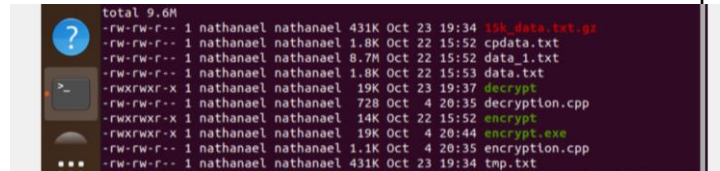


Then from there the data was added and committed to our public GitHub repository where

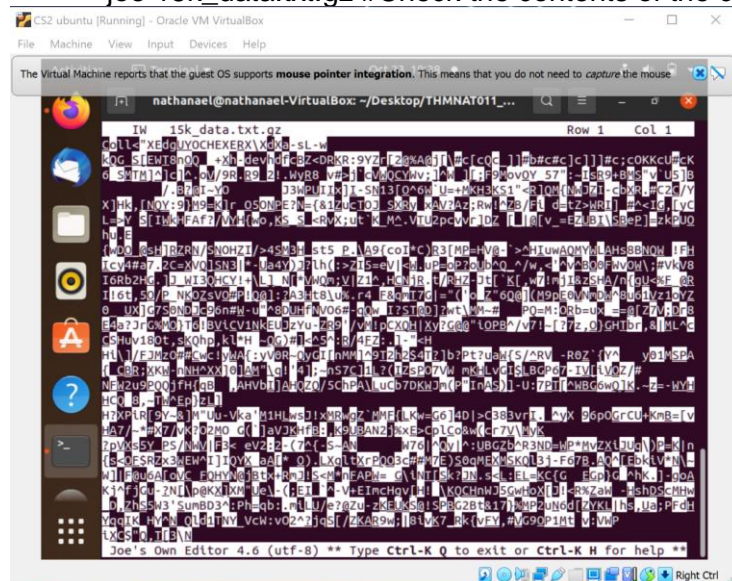
it will be pulled by a remote laptop running Ubuntu Linux.

These are the results that were collected from the laptop after decryption and decompression:

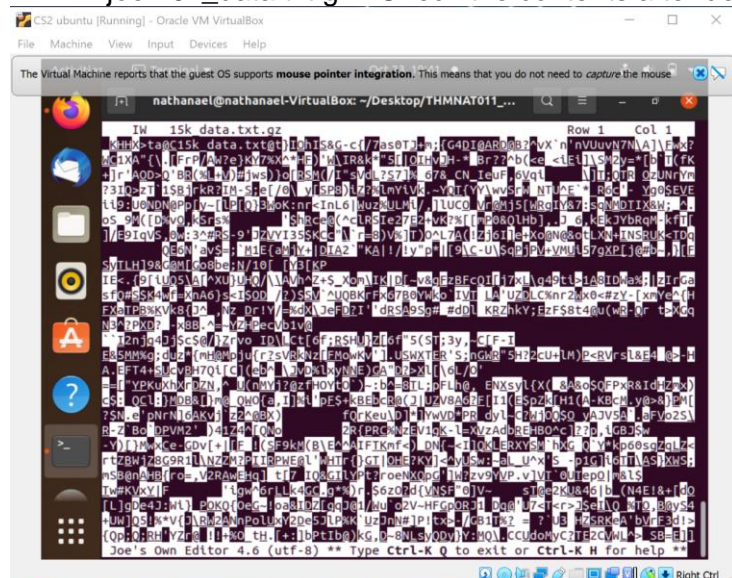
- `ls -lh` #Check the file is there and data is compressed.



- joe 15k\_data.txt.gz #Check the contents of the compressed and encrypted file



- ./decrypt #Decryption
- joe 15k\_data.txt.gz #Check the contents after decryption



- `gunzip 15k_data.txt.gz` #Decompression
- `ls -lh` #Check the size of the file after decompression

```

er$ ls -lh
total 11M
-rw-rw-r-- 1 nathanael nathanael 1.2M Oct 23 19:40 15k_data.txt
-rw-rw-r-- 1 nathanael nathanael 1.8K Oct 22 15:52 cpdata.txt
-rw-rw-r-- 1 nathanael nathanael 8.7M Oct 22 15:52 data_1.txt
-rw-rw-r-- 1 nathanael nathanael 1.8K Oct 22 15:53 data.txt
-rwxrwxr-x 1 nathanael nathanael 19K Oct 23 19:37 decrypt
-rw-rw-r-- 1 nathanael nathanael 728 Oct 4 20:35 decryption.cpp
-rwxrwxr-x 1 nathanael nathanael 14K Oct 22 15:52 encrypt
-rwxrwxr-x 1 nathanael nathanael 19K Oct 4 20:44 encrypt.exe
-rw-rw-r-- 1 nathanael nathanael 1.1K Oct 4 20:35 encryption.cpp
-rw-rw-r-- 1 nathanael nathanael 431K Oct 23 19:34 tmp.txt

```

- joe 15k\_data.txt #Check the contents of the file

| ID            | 15k_data.txt   | pitch         | yaw | AccX | AccY  | AccZ | Row 1 | Col 1 | GyrX |
|---------------|----------------|---------------|-----|------|-------|------|-------|-------|------|
| 1.08489258977 | 0.211823424482 | 176.37071092  | -1  | 235  | 16708 | 1    |       |       |      |
| 1.78259284741 | 0.375423336764 | 173.348115668 | -26 | 218  | 16800 | 1    |       |       |      |
| 2.26520518834 | 0.51012333154  | 170.421636034 | -32 | 250  | 16817 | 0    |       |       |      |
| 2.45625483357 | 0.567308373864 | 168.05737052  | -31 | 237  | 16798 | 0    |       |       |      |
| 2.45442301005 | 0.573877000906 | 166.325554022 | -36 | 222  | 16803 | 1    |       |       |      |
| 2.3590535182  | 0.555816846581 | 164.891078265 | -36 | 222  | 16799 | 1    |       |       |      |
| 2.26723486296 | 0.536990185609 | 163.567534146 | -29 | 225  | 16826 | 0    |       |       |      |
| 2.12512118566 | 0.498617420084 | 162.597638658 | -30 | 223  | 16820 | 0    |       |       |      |
| 1.94681078636 | 0.44328625253  | 161.974907987 | -22 | 218  | 16802 | 0    |       |       |      |
| 1.72180280578 | 0.379055120731 | 161.750768694 | -27 | 203  | 16807 | 0    |       |       |      |
| 1.57510011066 | 0.348255131487 | 161.377147162 | -42 | 212  | 16810 | 0    |       |       |      |
| 1.49992392925 | 0.32809290716  | 160.872213486 | -32 | 228  | 16812 | 0    |       |       |      |
| 1.36289847853 | 0.305983544365 | 160.686092963 | -52 | 201  | 16810 | 0    |       |       |      |

- Compare the contents of the original and extracted files

```

Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Adarsh> if(($?((Get-FileHash "C:\Users\Adarsh\Desktop\Git\THMNAT011_CRTBRA002_EEE3097S\check\15k_data.txt").hash -eq (Get-FileHash "C:\Users\Adarsh\Desktop\Git\THMNAT011_CRTBRA002_EEE3097S\check\15k_data.txt").hash)) {"Both files are identical"} else {"Both files are NOT identical"}
Both files are identical
PS C:\Users\Adarsh>

```

#### 5.4.2. Compression [CRTBRA002]

To avoid the repetition of results this section will be left out, therefore refer to Section 4.4.2 for the independent testing of the compression algorithm. This conclusion was made because the compression algorithm yields the same results for data sets of varying sizes and this can be seen from the system results derived in Sections 4.4.1 and 5.4.1.

#### 5.4.3. Encryption [THMNAT011]

To avoid the repetition of results this section will be left out, therefore refer to Section 4.4.3 for the independent testing of the encryption algorithm. This conclusion was made because the encryption algorithm yields the same results for data sets of varying sizes and this can be seen from the system results derived in Sections 4.4.1 and 5.4.1.

## 6. Consolidation of ATPs and Future Plan

### 6.1. Analysis of ATPs

| Description                                 | Acceptable                                                    | Not Acceptable                                              | ATP Met [Y/N] | Comment                                                                                                                                                                                                                      |
|---------------------------------------------|---------------------------------------------------------------|-------------------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Amount of Fourier coefficients extracted    | More than 25%                                                 | Less than 25%                                               | Y             | The original file is identical to the extracted file (according to the windows powershell). This validates that 100% of the Fourier coefficients are available because there is no change to the data in the extracted file. |
| Size of data set after compression          | Data set is reduced to 50% or more of original size           | Data set is not reduced to 50% or more of the original size | Y             | The original size of the file containing the data was 1.2Mb after gzipping the file thus compressing it the size of the file was 431 kb. This shows the the file has been reduced to 35.91% of its original size             |
| Integrity of the data set after encryption  | Outputted data set values are the same as the original values | Outputted data set values differ from the original values   | Y             | The integrity of the data is still there as after checking using the windows powershell it is seen that both files are identical.                                                                                            |
| Integrity of the data set after compression | Data set values remain the same as original values            | Data set values differ from the original values             | Y             | From the figures showing the windows powershell. With the result being that the files are identical.                                                                                                                         |
| CPU consumption                             | CPU usage is less than 50%                                    | CPU usage is more than 50%                                  | Y             | As can be seen in the figures above the average CPU usage in % barely went above 1.5 and the max when viewing one of the text files resulted in the average CPU usage to reach 2.05.                                         |

Table 5: Demonstrates if acceptance test procedures have been met or not.

## 6.2. Change to Specifications

The system functioned as expected and the results demonstrated that the specifications and requirements were met, therefore it was determined that there was no need to change the specifications set out in the previous report.

## 6.3. Future Plan

Some factors we would have to take into consideration when implementing our system in the real-world system (a SHARC Buoy):



- We would need to figure out a method that will automate the entire system as it will be placed in the SHARC buoy in a remote location in the Antarctic, and won't be able to be accessed directly using a wired connection.
- A replacement of the GitHub repository to something that makes use of the Iridium satellite networks.
- As our Raspberry Pi Zero W and Sense-HAT B are not waterproof and cannot operate in extreme cold, we would need to work on creating a casing that can protect it from water damage and the ice cold Antarctic temperatures.
- Looking for an external power source that will be able to last for extended periods of time as our system will be left for months on end, before the power source can be replaced.
- Looking for a better sensor HAT and processing unit that would be able to function at temperatures below zero.

## 7. Conclusion

The driving factor behind the compression of our data was because sending large data files would result in higher power consumption, and transferring data using Iridium satellite networks is very expensive. Because the data is also supposed to be confidential, the encryption proved to be an important component of the design. Only people who had access to a key would be able to decrypt and read the data.

The algorithm that was chosen to execute encryption and decryption was the AES algorithm and this allowed for easy implementation and design due to it being a relatively simple algorithm to design. LZ77 and Huffman coding was chosen for compression and decompression, using the gzip command to execute it. It proved to have the best computation time when compared to other algorithms. Because this implementation was already available as a command on both the Raspberry Pi Zero W and Linux based system it proved to work efficiently and effectively. No data was lost during compression or decompression. The algorithms also follow lossless compression and this made it easy to determine how many Fourier coefficients were made available after compression and decompression. From the results, it can be seen that the file before the compression, encryption, decryption and decompression and the file after this process were identical. This shows that the algorithms chosen didn't result in any of the data being changed.

The reasons behind why we chose C++ as the language of choice, was because it was the fastest and most efficient when compared to other languages (algorithms were able to execute in less than 3 seconds). Whereas, languages such as python took more than 30 mins to execute the encryption and compression. This would not allow us to meet our requirement concerning minimal computation so as to not result in heavy power consumption.

All the ATPs were met, and system requirement and specifications were implemented as stated in the allocated sections above.

### Recommendations:

- Run software-based and hardware-based simulations in varying temperature conditions to identify the operating range of the system.

- Attach a the device to a buoy in a pool so as to simulate some of the conditions that the real-world SHARC buoys would experience, this would make data generation more accurate.
- Automate file generation, compression and encryption so as to better emulate the real-world situation (the SHARC Buoy will be in a remote location and can't be accessed directly).

## 8. References

- [1] Codescracker.com. 2021. *C++ Program to Encrypt and Decrypt a File*. [online] Available at: <<https://codescracker.com/cpp/program/cpp-program-encrypt-file.htm>> [Accessed 5 October 2021].
- [2] B. Carthew and N. Thomas, "Submission 2: First Progress Report," Oct. 2021.
- [3] Waveshare.com. 2021. *Sense HAT (B) - Waveshare Wiki*. [online] Available at: <[https://www.waveshare.com/wiki/Sense\\_HAT\\_\(B\)](https://www.waveshare.com/wiki/Sense_HAT_(B))> [Accessed 24 October 2021].
- [4] Product.tdk.com. 2021. *World's First Wide-Range 6-Axis MEMS MotionTracking™ Device for Sports and High Impact Applications*. [online] Available at: <[https://product.tdk.com/system/files/dam/doc/product/sensor/motion-inertial/imu/data\\_sheet/ds-000192-icm-20649-v1.0.pdf](https://product.tdk.com/system/files/dam/doc/product/sensor/motion-inertial/imu/data_sheet/ds-000192-icm-20649-v1.0.pdf)> [Accessed 24 October 2021].
- [5] J. Thakkar, "Types of Encryption: 5 Encryption Algorithms & How to Choose the Right One", *Hashed Out by The SSL Store™*, 2021. [Online]. Available: <https://www.thesslstore.com/blog/types-of-encryption-encryption-algorithms-how-to-choose-the-right-one/>. [Accessed: 03- Sep- 2021].
- [6] "Lossy compression and Lossless compression algorithms", *PeaZip file archiver utility, free RAR ZIP software*, 2021. [Online]. Available: <https://peazip.github.io/lossless-compression-lossy-compression.html>. [Accessed: 03- Sep- 2021].
- [7] "Lossy vs Lossless | Differences, Types, Benefits & Examples", *Teach Computer Science*, 2021. [Online]. Available: <https://teachcomputerscience.com/lossy-vs-lossless/>. [Accessed: 03- Sep- 2021].
- [8] "Difference Between Lossy Compression and Lossless Compression", *BYJUS*, 2021. [Online]. Available: <https://byjus.com/gate/difference-between-lossy-compression-and-lossless-compression/>. [Accessed: 05- Sep- 2021].
- [9] S. Khayam, "The Discrete Cosine Transform(DCT): Theory and Application", *Egr.msu.edu*, 2003. [Online]. Available: [https://www.egr.msu.edu/waves/people/Ali\\_files/DCT\\_TR802.pdf](https://www.egr.msu.edu/waves/people/Ali_files/DCT_TR802.pdf). [Accessed: 03- Sep- 2021].
- [10] M. Nelson, "Data Compression With Arithmetic Coding", *Mark Nelson*, 2014. [Online]. Available: <https://marknelson.us/posts/2014/10/19/data-compression-with-arithmetic-coding.html>. [Accessed: 03- Sep- 2021].
- [11] "Huffman Coding", *Tutorialspoint.com*. [Online]. Available: <https://www.tutorialspoint.com/huffman-coding>. [Accessed: 05- Sep- 2021].
- [12] I. News and I. News, "5 Common Encryption Algorithms and the Unbreakables of the Future", *StorageCraft Technology, LLC*, 2021. [Online]. Available: <https://blog.storagecraft.com/5-common-encryption-algorithms/>. [Accessed: 05- Sep- 2021].