University of Cape Town

EEE3097S

Engineering Design: Electrical And Computer Engineering

Final Report

Authors:
David Young
Caide Spriestersbach

Student Numbers:
YNGDAV005
SPRCAI002

October 2022

Table of Contents

Table of Figures

List of Tables

A. Administrative Details

i. Individual Contributions

Both members worked equally on the project and contributed to each section. However, some areas were focused more heavily on by a specific member. These sections are listed below.

Table I – Table of individual contributions made by each member

Name	Description	Sections	Pages
David Young YNGDAV005	Equally contributed to Paper Design. Major contributions were on experiment designs, results, and ATPs throughout Progress Report 1 and Progress Report 2.	C.1 C.3 C.5 D.1.1.4 D.1.1.6 D.2 E.2.1 E.2.2 E.2.4 E.3.1 E.3.2 E.3.4 E.4.1 E.4.3 F.1.3, F.2.1 F.2.2 F.2.4 F.2.5 F.3.1 F.3.2 F.3.4 F.3.5 F.4.1 F.4.3 F.4.4 J	6, 8, 9, 10, 13, 14, 18, 19, 20, 26, 27, 28, 29, 31, 32, 33, 35, 36, 37, 28, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 62, 63, 64, 65
Caide Spriestersbach SPRCAI002	Equally contributed to Paper Design. Other contributions were on Data usage, experiment design, ATPs, and IMU details throughout Progress Report 1 and Progress Report 2. Majorly contributed to consolidation of ATPs, Future Plans, and Conclusion	B C.1 C.2 C.4 D.1.1.1 D.1.1.2 D.1.1.3 D.1.1.5 D.1.2 D.1.3 E.1 E.2.3 E.3.3 E.4.2 F.1.1 F.1.2 F.2.3 F.3.3 F.4.2 F.4.4 G.1.1 G.1.2 G.1.3 H	5, 6, 7, 9, 11, 12, 13, 14, 15, 16, 17, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 33, 34, 35, 39, 40, 47, 48, 49, 56, 57, 58, 59, 60

ii. Project Management Tool

Below is a screenshot of the front page of our project management tool as of Thursday, 13 October 2022.

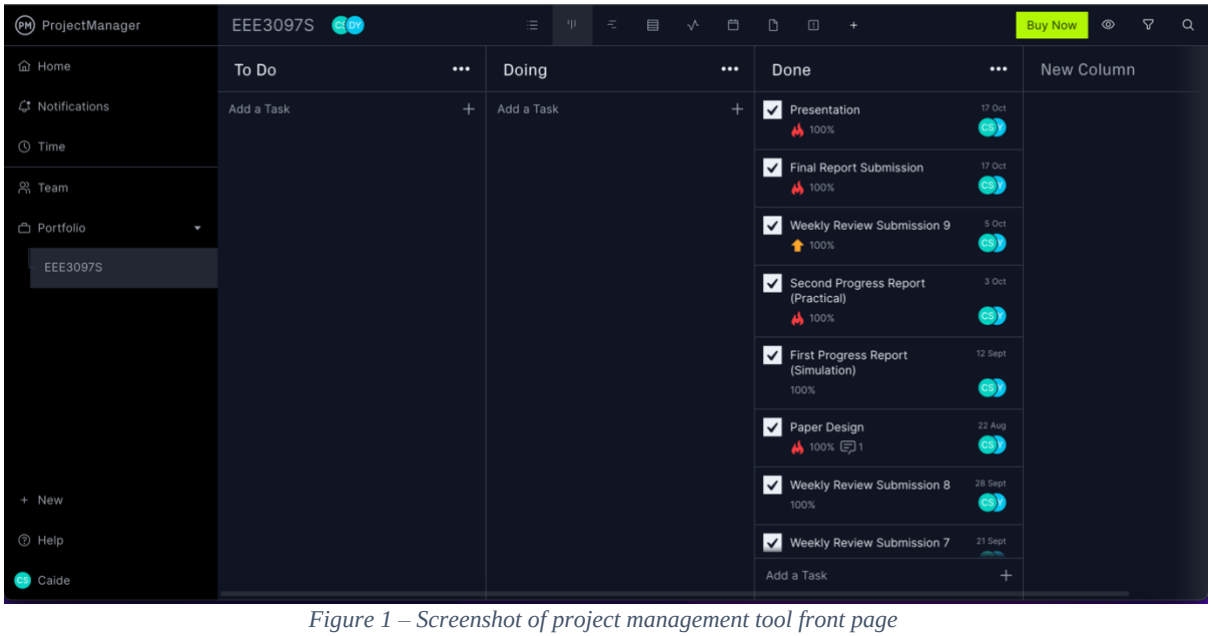

Figure 1 – Screenshot of project management tool front page

Link to project manager tool.

iii. Development Timeline

As can be seen by the timeline below, the development is on time.

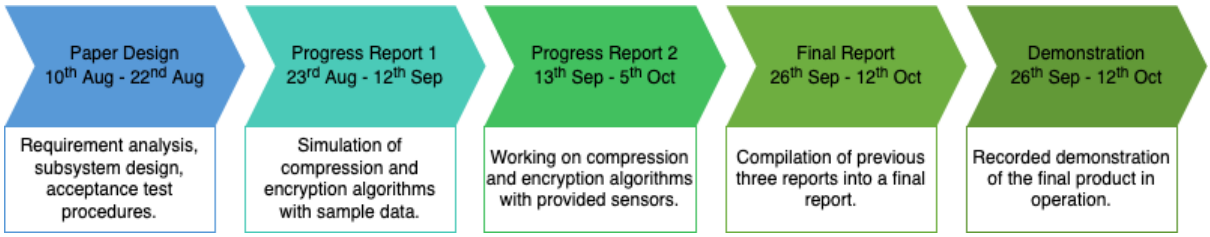

Figure 2 – Development timeline for the project

iv. GitHub Link

The members made use of a GitHub git repository used throughout this project. Below is a link to the GitHub repository.
https://github.com/the-user-created/EEE3097S-Project

B. Introduction

As a part of studying the movement and atmosphere of the Antarctic pancake ice, the SHARC buoy in Antarctica consists of multiple sensors and other processors. The following report will outline the requirements and design procedure for the compression and encryption of data received by the Inertial Measurement Unit (IMU) – one of the crucial sensors onboard the buoy. The data is compressed to reduce the transmission cost, as Iridium is exceptionally costly.

The compression and encryption will be done on the STM32F051 Discovery Board, which will be connected to the Sense HAT B – which houses the sensors and records the data. The STM32 Discovery Board makes use of the ARM Cortex M0 core. This will be responsible for processing, encrypting, compressing, and transmitting the relevant data packets from the IMU, which will be used to analyse the environment surrounding Antarctica.

The report will analyse the requirements outlined in the project design brief, present the subsystem and the sub-subsystem design, outline the acceptance test procedures, and the development timeline of the overall design. In doing so, the report will summarise each requirement derived from the project brief, compare various compression and encryption algorithms to determine the most suitable algorithm, analyse the feasibility of the overall project, present possible bottlenecks of the system, outline Subsystem and Sub-subsystem requirements and specifications, as well as the Inter-Subsystem and Inter-Sub-subsystems interactions, present the UML diagrams for the overall system. The acceptance test procedures will make use of figures of merit, which will be used to validate the design. The report will present and summarise the experiment design used to test the figures of merit and adumbrate the acceptable performance definition.

C. Requirement Analysis

1. Interpretation of the requirements

The following is extrapolated based on the requirements outlined in the design project brief. The requirements below will be used to produce the functional requirements and hence the specifications in the subsequent sections. These requirements are listed in no particular order.

Table II – Interpretation of user requirements

User Requirement ID	Description
UR1	Data must be encrypted.
UR2	Data must be compressed.
UR3	Client requires at least 25% of the lower Fourier coefficients of the data.
UR4	Power usage must be low to increase battery life.
UR5	Software may be written in any language.
UR6	Processing load on the STM board must be low.
UR7	A motion tracking device must be used to collect the data.
UR8	The STM32F051 Discovery Board must be used.

2. Comparison of encryption algorithms

Encryption can be broken down into two types, asymmetric and symmetric. Symmetric cryptography uses the same key for encryption and decryption [1]. A unique key is produced for each pair of participants [1]. Since symmetric encryption uses keys that are only known to the two parties making use of the encryption, this brings about a level of authentication. The benefits of using symmetric cryptography are that it is relatively inexpensive for a strong key for ciphering to be produced, the algorithms are computationally cheap, the level of protection dictates the size of the key, and the keys tend to be smaller for the level they afford [1]. The possible downfalls of symmetric encryption are that the privacy of the key must be retained to ensure confidentiality and authentication – this usually means that the key must be encrypted in another key, and the recipient must already have the key for decryption of the encrypted secret-key [1].

Asymmetric cryptography (also known as public-key encryption) uses a pair of keys, known as the public and private keys, which are associated with an individual who needs to encrypt data [1]. Each public key is published while their corresponding private keys are kept secret

[1]. All data encrypted with the public key can only be decrypted using the corresponding private key [1]. The possible benefit of using an asymmetric algorithm is that the transmission of data packets will be protected in a public domain environment, i.e., the internet [1]. The possible downfalls are that if the private key becomes lost or forgotten, the sent data cannot be decrypted, as authentication cannot be made.

The Advanced Encryption Standard (AES) is a symmetric cryptographic algorithm that can be used to protect data [2]. The AES is a block cypher, meaning it will take in data of a fixed size in bits and produce the exact size of bits of ciphertext – an unintelligible form of data [3]. The algorithm has the capacity of using keys of lengths 128, 192, and 256 bits to encrypt and decrypt blocks of data of size 128 bits. The AES is highly efficient for 128-bit form, can be implemented in hardware and software, has open source code, and is invulnerable to most attacks except brute force.

The RSA algorithm is an asymmetric cryptography algorithm. It has become the standard for encrypting and decrypting data shared via the internet. The algorithm's encryption strength depends on the key's size, and a larger key size means a more secure algorithm. The algorithm is known to be robust and reliable but also known to be computationally intensive.

Blowfish is a symmetric block cypher cryptography algorithm. The most notable features of this algorithm are that the cypher block size is 64 bit, variable key length from 32 to 448 bits, known to be much faster than other algorithms, royalty-free (in the public domain), and no license required to use [4]. The algorithm is a quick, reliable, flexible, and unbreakable [4]. The biggest downfall is the 64-bit block, which the algorithm encrypts individually and is not as secure as other algorithms.

Twofish is the predecessor to blowfish and is the same as blowfish, with the difference that it is faster, more flexible, and more secure. It, too, is license free and in the public domain. It has a 128-bit data block, encrypts in 16 rounds, and is ideal for hardware. It was the fastest algorithm across all CPUs, competing to become the AES, and it fits into a few gates in the hardware [5]. There is no other algorithm available with the same flexibility in implementation, the ability of Twofish to trade off key-setup time and RAM and ROM for encryption speed [5].

Although AES is widely used and open-sourced, the code is not as small in size and length as Twofish. Twofish has the same benefits as AES but has shorter and faster code. RSA is asymmetric, which is not the preferred method of key encryption as it will be computationally intensive, which goes against UR006. Blowfish is faster than most algorithms but is not as secure as Twofish, while it does share the same benefits. The two encryption algorithm has a unique combination of flexibility, speed, and conservative design [5].

Although AES is widely used and open-sourced, the code is not as small in size and length as Twofish. Twofish has the same benefits as AES but has shorter and faster code. RSA is asymmetric, which is not the preferred method of key encryption as it will be computationally intensive, which goes against UR006. Blowfish is faster than most algorithms but is not as secure as Twofish, while it does share the same benefits. The two encryption algorithm has a unique combination of flexibility, speed, and conservative design [5].

3. Comparison of compression algorithms

Compression can be either lossy or lossless. In lossless compression, the physical file size is reduced while the integrity of the data is retained. This means that the original data can be perfectly recovered at the decompression stage without any depreciation of data quality. In lossy compression, the compression stage comprises the data quality by eliminating data which is not noticeable. This means that the original data can never be recovered after the compression is complete. Lossy compression compresses images, audio, and video, whereas lossless compression compresses text, audio, and images [6].

One of the foundational compression algorithms, LZ77, was created by Abraham Lempel and Jakob Ziv in 1977 [7]. Some notable compression software used today that are based on LZ77 are ZIP and GZIP [8]. The main idea behind LZ77 is to replace multiple occurrences of the same sequence of bytes with an index reference to the first occurrence of that sequence. This is achieved by using a sliding window to inspect the source sequence of bytes and to maintain a ledger of historical data that serves as a dictionary [9]. The sliding window comprises of a look-ahead buffer and a search buffer. The search buffer consists of the dictionary and the recently encoded data. The lookahead buffer contains the next portion of the input data sequence to be encoded [9]. The dictionary contains three values; Offset (distance between the start of sequence and beginning of the file), run-length (number of characters in the sequence), and the deviating characters (markers indicating a new sequence) [7]. The sliding window size is one of the major factors which affects the performance of the compression [9]. The shorter the sliding window size is, the faster the compression will take. However, the likelihood of finding repeated sequence occurrences will decrease; hence, the resultant compressed file will increase in size. Therefore, there is a trade-off to be made between the speed of compression and the effectiveness of the compression. In practice, the effectiveness of the compression is correlated to the data being compressed, and the sliding window size can be from several kilobytes to several megabytes [9].

LZR is a modification of LZ77 that was invented by Michael Rodeh et al. in 1981 [10]. LZR was designed to be a linear time alternative to LZ77; however, the encoded pointers can point to any offset in the file [7]. This results in the memory requirements of LZR increasing as the size of the input string increases [10]. Combined with its poor compression ratio, LZ77 is often superior; LZR is an impractical variant.

Lempel-Ziv-Storer-Szymanski [11], or LZSS, is another algorithm designed as a variant of LZ77. It was first published in 1982 by James Storer and Thomas Szymanski. LZSS manages to improve upon LZ77 by detecting whether a given substitution will decrease the compressed file size or not [7]. If it doesn't, the input is left in its original form; otherwise, the sequence is substituted with an offset-length pair [7]. LZSS also improves upon LZ77 by eliminating the use of deviating characters. This decreases the size of the dictionary and hence improves the efficiency of the compression. LZSS is commonly used for archive formats such as RAR [7].

Deflate (stylised as DEFLATE) was invented by Phil Katz in 1993 [12] and has become the basis for most compression tasks today. Deflate combines the LZ77 (or LZSS pre-processor) algorithm with Huffman coding. Huffman coding is a lossless entropy (smallest number of bits needed, on average, to represent a symbol) encoding compression method which assigns codes based on the frequency of a symbol occurrence [7]. The codes are stored in a Code Book,

constructed for the input string. Each of these codes is variable-length, where the length of the code is based on the frequency of the corresponding symbols [9]. There are two main properties behind Huffman coding; codes for more frequently occurring symbols are shorter than codes for less regularly occurring characters, and each code can be uniquely decoded [9]. Due to the last property, each of the codes must be prefix codes, meaning that the code for a symbol cannot be a prefix of a code for any other character. Huffman encoding has two steps. The first step is to build a Huffman tree (type of binary tree) of the original symbols from the input string. Then the Huffman tree is traversed, and codes are assigned to each character in the tree.

The final lossless compression algorithm to be evaluated is Brotli. Brotli is developed by Google. Brotli uses both LZ77 and Huffman coding like Deflate but introduces 2nd order context modelling to increase the effectiveness of the compression [13] while retaining similar speeds to Deflate. Context modelling uses preceding events to model the next event.

4. Feasibility analysis

The microcontroller uses a fixed-point ARM processor, which means it cannot process floating point values. This may affect the compression and encryption algorithms, leading to readdressing the choice of the algorithm used.

The algorithm used, either for compression or encryption, could be too computationally intensive for the microcontroller – which may result in a revision in which a slower but less computationally intensive algorithm must be used. The algorithm for compression may have a high loss rate, failing to meet the requirement UR003.

The IMU has different modes of operation, which will allow for a low power consumption mode, increasing the battery's life span. Other operation modes could lead to a short life span of the battery, which could hinder the entire system.

Another issue which could hinder the project is that the Discovery Board only has 64kb flash memory and 8 bytes of RAM, which means that a complex method will be needed to run the compression and encryption algorithm on the STM Discovery Board itself.

Filtering the data so that 25% of the IMU's coefficients are extracted may be a challenge to implement. It is also tough to process and work with the STM Discovery Board due to our lack of experience and knowledge of the hardware. Further research and learning may be required.

Despite the above challenges, we have the time and resources to ensure that all user requirements are met.

5. Possible bottlenecks

Some trade-offs and restrictions have already been detailed in the sections above. Some are yet to be documented. The bottlenecks due to trade-offs expected to occur are described in the following paragraphs.

In terms of data collection, the size of each packet of data that needs to be transferred must be considered. The data link between the STM discovery board and the sensor HAT could create congestion if the transfer speed is too slow or the size of packets is too large.

The next bottleneck to consider is the processing time. This bottleneck is affected by the collection of data bottleneck detailed above. The larger the packet of data to be compressed and encrypted, the longer it will take to be processed and the longer it will take to be transmitted.

The STM storage is limited to 64 kilobytes of flash memory. Initial estimations place the size of the software around 34 kilobytes. This means there are only 30 kilobytes of 'free' storage for data collection before compression and encryption.

The choice of compression and encryption algorithms could become a significant issue. The stronger the compression, the longer it will take to compress a given file, as well as increasing the amount of RAM used for the compression software. In terms of encryption, every encryption algorithm will operate at different speeds depending on the input data type. Some algorithms work better with finite data lengths, whereas others work better with varying data lengths.

The collection, compression or data encryption processing time may require significant data processing time. In that case, the power usage of the STM will increase significantly. Depending on the physical set-up of the buoy-STM system, this could be either a minor or major issue.

D. Paper Design

1. Subsystem Design

1.1 Subsystem and Sub-subsystems Requirements and Specifications

1.1.1 Retrieval & Storage of Data

This subsystem focuses on retrieving the data generated by the IMU and storing said data on the STM32F0 Discovery board (further referred to as STM). We are constrained only to use 64 kilobytes of flash memory and 8 kilobytes of RAM due to the design of the STM. The STM cannot add external storage space like an SD card.

Table III – Functional requirements of the retrieval and storage of data

Functional Requirement ID	Description	User Requirement Addressed
FR1	The IMU is a 9-axis MEMS Motion Tracking device is used.	UR7
FR2	Data cannot be stored in large quantities on the STM.	UR8
FR3	The processing on the STM must not be computationally intensive.	UR6

Table IV – Design specifications for the retrieval and storage of data.

Design Specification ID	Description	Functional Requirement ID
DS1	The ICM-20948 MEMS motion tracking is used.	FR1

1.1.2 Data Processing

UR003 states that at least 25% of the lower Fourier Coefficients must be extracted. This reduces the strain on the STM as it has limited onboard storage, and as there is no possibility of increasing that storage, this will play a vital role in the entire system. The reduction in the size of the collected data will also increase the efficiency of the compression and encryption algorithm. The IMU cannot record the data in the frequency domain; it measures in the time domain – it is essential for a system to be designed to transform the data to the frequency domain. It is desired to use the Fast Fourier Transform (FFT) to do such a transform for efficiency. To use the FFT, the correct sample time must be used, and the amount of data collected must also be compatible with the onboard storage of the STM. This will allow the subsequent Fourier coefficients to be collected and processed. Based on research from the STM IDE page, it is recommended that the programming language to be used is either C, C++,

Pascal, or Java. In the case of this project, we will use a combination of programming languages on the computer but predominantly use C on the STM.

Table V – Functional requirements for data processing

Functional Requirement ID	Description	User Requirement Addressed
FR4	Some of the Fourier coefficients must be retained after compression	UR3
FR5	The programming language used must be compatible with the STM Discovery Board without modifications.	UR8 & UR5

Table VI – Design specifications for data processing

Design Specification ID	Description	Functional Requirement ID
DS2	At least 25% of the Fourier Coefficients must be retained, this can be done by filtering the data using code.	FR4 & FR5

1.1.3 Encryption of data

The encryption will be done using the Twofish algorithm. This is one of the world's fastest, most flexible and most secure algorithms. Twofish is also compatible and easy to implement on the STM32F051 Discovery Board. Ideally, the encryption will be run on the STM, but this depends on the onboard memory. Encrypting data packets that are transmitted is good practice for ensuring the confidentiality of the information. Twofish will provide secure encryption for the data from the SHARC buoy to the researchers.

Table VII – Functional requirements for encryption of the data

Functional Requirement ID	Description	User Requirement Addressed
FR6	All data received by and from the STM must be suitably encrypted.	UR1
FR7	The encryption algorithm must be light and computationally inexpensive.	UR6 & UR4

Table VIII – Design specifications for encryption of the data

Design Specification ID	Description	Functional Requirement ID
DS3	Twofish algorithm to be used as it is lightweight and is suitable for all types of encryption.	FR6 & FR7

1.1.4 Compression of Data

Based on the comparisons made in section C, the requirement analysis, between various compression algorithms, the LZSS compression algorithm was chosen. This algorithm was selected due to its high compression ratio, speed, and relative lack of complexity. The data compression will occur before the data's encryption. This is because, after encryption, the data becomes primarily random; hence, the compression algorithm is unlikely to find enough matching sequences of characters to compress the data file effectively.

The SHARC buoy requires a compressed file due to the use of a satellite link to transmit the data. This method of communication has high data transfer costs and inconsistent transmission and bandwidth. Therefore, any efforts to reduce the transmission file size are critical.

Table IX – Functional requirements for compression of data

Functional Requirement ID	Description	User Requirement Addressed
FR8	All data received by and from the STM must be suitably compressed.	UR2
FR9	The compression algorithm must be effective and computationally inexpensive.	UR6 & UR4

Table X – Design specifications for compression of data

Design Specification ID	Description	Functional Requirement ID
DS4	LZSS algorithm is to be used as it is effective and is suitable for text based compression.	FR8 & FR9

1.1.5 Transmission of Data

Once the data has been compressed, encrypted, and checked if the contents are sufficient – have at least 25% of the lower Fourier coefficients, the data will be transmitted from the SHARC buoy via Iridium. Iridium is a satellite communications company whose only service is providing voice and data services to everywhere on the earth, even the remote Antarctic. Iridium is extremely costly, so compression and encryption are vital in ensuring the project's feasibility.

Table XI – Functional requirements for the transmission of data

Functional Requirement ID	Description	User Requirement Addressed
FR9	All compressed and encrypted data must be received by the researchers.	UR1 & UR2 & UR3

Table XII – Design specifications for the transmission of data

Design Specification ID	Description	Functional Requirement ID
DS5	All encrypted and compressed data will be transmitted via the Iridium network from SHARC buoy to researchers.	FR9

1.1.6 Checksum

Before the compression and encryption of the data, a CRC-32 checksum of the data file will be created. After the data is transmitted, decrypted, and decompressed on the client side, another CRC-32 checksum will be made. These two checksums can then be compared to verify whether the data file has changed during any of the stages between the data collection and the data decompression.

Table XIII – Functional requirements for the checksum subsystem

Functional Requirement ID	Description	User Requirement Addressed
FR10	The data received by the researchers must be identical to the data recorded on the STM.	UR3

Table XIV – Design specifications for the checksum subsystem

Design Specification ID	Description	Functional Requirement ID
DS6	The integrity of the data will be verified by a CRC-32 checksum.	FR10

1.2 Inter-Subsystem and Inter-Sub-subsystems Interactions

All the above submodules form the primary system where the compression, encryption and transmission of the data received from the IMU will occur. Although the STM itself is a submodule of the overall SHARC buoy, this paper design treats the STM as if it were the main module – all of the above submodules form a part of the larger STM submodule. The IMU-20948 is another submodule apart from the SHARC buoy, which will interact with the STM, allowing for the transmission of data from the sensor to the STM. This data transmission between the IMU and STM is a sub-subsystem of the overall buoy system.

Once the STM has filtered the data, a CRC-32 checksum of the data file will be created. The data will be compressed using the LZSS algorithm. This task is performed by the 4th submodule outlined above. Following compression, the Twofish algorithm encrypts the data before it is transmitted to the computer – this is the 3rd submodule described above. Following encryption, the data processing is complete, and the encrypted data packet is now ready for transmission via Iridium to the researchers.

It should be noted that the entire subsystem depends on the lifespan, battery life, and storage on the STM Discovery board and is susceptible to the IMU-20948, which will determine the sample rate. This dictates the efficiency of submodule one above and will impact the overall SHARC buoy. The processes described above are explained in the diagram below.

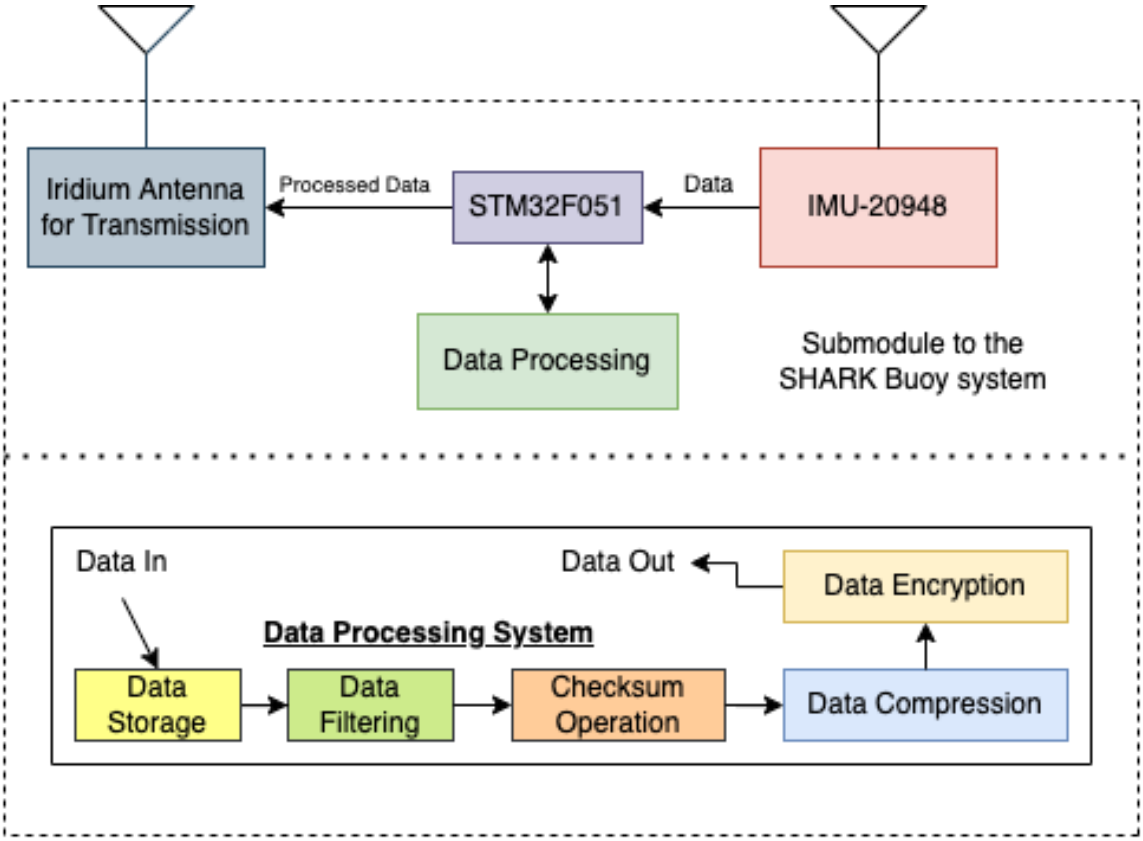

Figure 3 – Diagram of the system in operation

1.3 UML or OP Diagrams

The following UML diagram depicts how all the submodules interact to make up the overall SHARC buoy system and the flow in which the submodules operate. Note that this is a sequential process – a submodule must complete its operation before the next submodule can proceed with its task.

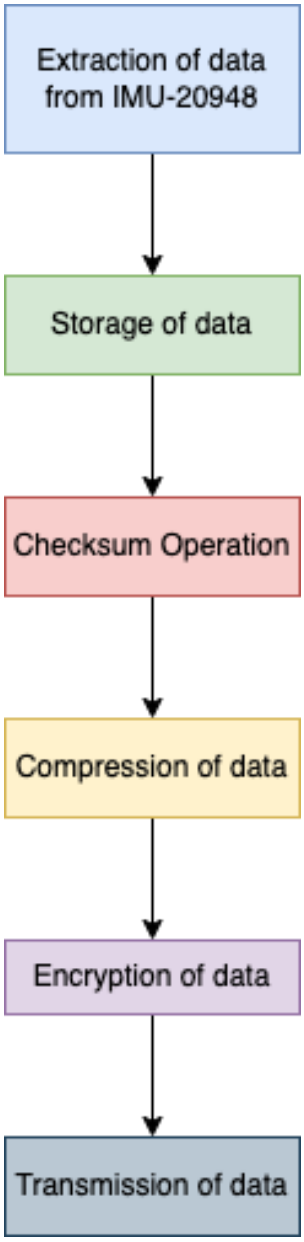

Figure 4 – Diagram showing the operation of the STM module of the System

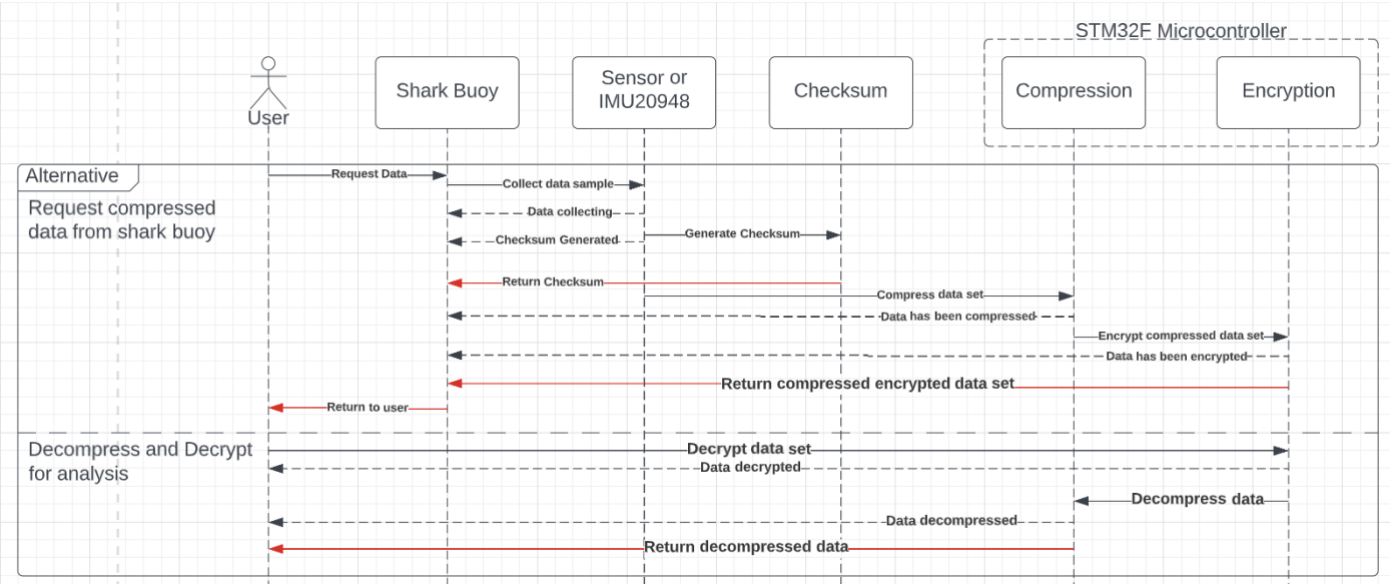

Figure 5 – UML Sequence Diagram Displaying the use case scenario

It should be noted here that the STM used on the SHARC buoy is the same STM microcontroller used for the decryption and decompression of the data set which is returned by the SHARC buoy. This is done to ensure the same ARM processor and architecture which will guarantee that the data decompressed and encrypted is the same, this is verified using the checksum.

2. Acceptance Test Procedure

The acceptance test procedures used to evaluate the performance of the design and whether it meets the minimum requirements are analysed in the following subsections below.

2.1 Figures of Merit

Table XV – Figures of merit for each subsystem

Subsystem	Figures of Merit
1: Retrieval & Storage of Data	The sample time of sensor values should be correct. The sampled data should not exceed the storage restrictions due to the size of the code files.
2: Data Processing	Determine the execution speeds, efficiency, and correct processing of the STM.
3: Encryption of Data	The execution time of the algorithm should not exceed 10 seconds per 10 kilobytes of data. The algorithm should be secure as well as unlikely to be decrypted.
4: Compression of Data	The compression algorithm should be efficient and take no longer than 5 seconds per 10 kilobytes of data. No data must be lost during the compression of the data file. The compression ratio of the original file size and the compressed file size should be at least 1.5.
5: Checksum	The checksum of the original data file should be created successfully on the STM and it should match the checksum of the decompressed data file on the client-side.
6: Transmission of Data	No packet loss should occur during transmission of the encrypted data file.

2.2 Experiment Design of ATPs

The compression and encryption experiment design handle the data processing subsystem ATP. The following tests will be conducted using the computer in which is connected to the STM. The implications of running the tests on the computer rather than the STM is that during operation on the SHARC buoy it will not be possible to verify that the system is working the same as it was during testing. However, this can be overcome by storing the test results of the initial design testing on the computer and compared against the results of the STM onboard the SHARC buoy as a means of error detection and validation.

2.2.1 Experiment Design to Test the Compression ATPs

Various sample IMU datasets are available for testing the compression figures of merit. The LZSS compression algorithm will be used to compress the data file. The LZSS decompression algorithm will also be used to decompress the compressed file. The resultant compressed file can be compared to the original data file to determine the compression ratio. The decompressed file can then be compared to the original data file to determine whether the data was corrupted. The execution time of the compression (and decompression) algorithm can be recorded to create a benchmark to determine the mean speed of compression.

2.2.2 Experiment Design to Test the Encryption ATPs

Various sample IMU datasets are available for testing encryption figures of merit. The Twofish algorithm will be used to encrypt the data file and decrypt the encrypted file. The quality of the encryption algorithm can be determined by attempting to decrypt the file using different keys. If any key besides the key generated by the encryption algorithm manages to decrypt the file, then the encryption is not successful. The execution time of the encryption (and decryption) algorithm can be recorded to create a benchmark to determine the mean speed of encryption. The last test will compare the original file to the decrypted file to determine whether it is human-readable.

2.2.3 Experiment Design to Test the Checksum & Transmission of Data ATPs

A CRC-32 checksum of the data file can be created before compression and encryption. The data file will then be compressed, encrypted, and transmitted to the user. The file can then be decrypted and decompressed on the users' end, and a CRC-32 checksum of the resultant file can be created. The users' checksum and the STM's checksum can then be compared. If the two checksums are different, then the original data file was likely compromised during any stages between the creation of the two checksums.

2.3 Acceptable Performance Definition

The design needs to be able to compress the data sufficiently enough to be efficiently transferred by the Iridium satellite network. However, no data should be lost during the compression process; therefore, lossless compression must be used. The design also needs to encrypt the compressed data to ensure that the transmission is confidential. The processing handled on the buoy must consume as little battery power as possible to ensure that the design does not incur frequent battery recharging.

2.3.1 Compression subsystem

Table XVI – ATPs for the compression subsystem

ATP	Description
Compression time	The compression algorithm must take no longer than 5 seconds per 10 kilobytes of data. The system passes if it achieves this requirement.
Data loss	No data must be lost during the compression of the data file. The system passes if it achieves this requirement.
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5. The system passes if it achieves this requirement.

2.3.2 Encryption subsystem

Table XVII – ATPs for the encryption subsystem

ATP	Description
Encryption time	The execution time of the encryption and decryption should not exceed 10 seconds per 10 kilobytes of data. The system passes if it achieves this requirement.
Data loss	No data must be lost during the encryption and decryption of the data file. The system passes if it achieves this requirement.
Encryption security	The encryption must be strong enough to prevent trivial decryption. The system passes if it achieves this requirement.
Encryption integrity	The original data file's contents must be identical to the decrypted data file's contents. The system passes if it achieves this requirement.

2.3.3 Checksum subsystem

Table XVIII – ATPs for the checksum subsystem

ATP	Description
STM checksum	The STM must create a checksum of the original data file successfully. The system passes if it achieves this requirement.
Client checksum	The client's device must create a checksum of the decrypted and decompressed file successfully. The system passes if it achieves this requirement.
Checksum comparison	The client's and STM's checksums must be identical. The system passes if it achieves this requirement.

E. Validation using Simulated or Old Data

1. Data

Supplying the algorithms used for compression and encryption with simulated data was done to validate the workings of these algorithms. The following information concerns the data sets used, the file formats and how this relates to the compression and encryption for the SHARC Buoy project.

1.1. Data Used

The members decided to use simulated data as it would replicate the type of information we would receive from the Sense Hat B on the buoy in Antarctica. The file formats used in testing were binary, text and Comma-Separated Values (CSV), as it is common in Digital Signal Processing to use these file formats. Hence, it is essential to test the encryption and compression of these file types. Encryption and compression occurred on different data sets (in three file formats). The first data set was collected by the user carrying the Inertial Measurement Unit (IMU) in their pocket and walking around, sitting down and standing for over 5 minutes. The second data set recordings were from the IMU being fixed on a rotation platform – the IMU was fixed to one of its axes, and the data was recorded for 10 minutes. The third data set was the same recording environment as the second, however, at a lower sample rate. The team decided to use every data point available in the sample data set, this included all axes for the gyroscope, and accelerometer as well as the time, count, yaw, pitch, roll, and temperature as it was decided that the more data sets available now with the current working system will mean that if there needs to be additional data collected in future on the SHARC buoy, the system does not need to be modified to accommodate these changes; allows for scalability. This data also allows for the testing of the compression time, whether data is lost through the system, and if the compression ratio is desired as means of determining if the ATPs for compression have been met. Likewise, this testing with simulated data will allow the same testing for the encryption time, whether the encryption is secure, and if the file passed to the system is identical to the decrypted and decompressed file; this will also verify the ATPs for the checksum mainly a checksum is generated and the checksum being identical at the beginning and end of testing.

1.2. Justification of Data Used

The simulated data will accurately represent the data recorded on the buoy in Antarctica; therefore, the data sets will be suitable for testing the compression and encryption blocks since we were unable to collect data from a sensor for testing; it is important to use simulated data to ensure the functionality of the system. Different file sizes and types were tested to test the efficiency of the encryption and compression algorithm and ensure functionality. The file format in which the data will be stored and processed from the Sense Hat and IMU has not been finalised. The file format to be used will be completed once testing with the IMU and

Sense Hat commences. It was then decided to test multiple file formats in the simulation phase to ensure that the encryption and compression would work with the file format used in the final product. The simulated data will be compared against the experimental data to test the acceptance test procedures and receive figures of merit. Following compression, encryption, decryption and decompression, the file will be in the same format as the IMU recorded; this can only be achieved in the simulation if the data set and format are the same as the one used in the practical application.

1.3. Initial Analysis of Data

Since the files contain similar data sets, it was decided to use the first thousand samples from the "EEE3097S 2022 Turntable Example Data 2.csv". The following three plots depict the relationship of the data collected from the IMU in the time domain; the following three plots show a sample of the Fourier Transform. Since the compression and encryption algorithms are both lossless, 25% of the Fourier Coefficients will be preserved – satisfying the user requirement. The project focuses on the encryption and compression of the files which contain the data; it was therefore not considered of high importance to value the IMU's actual data or actual readings. The following is however an example of how the data collected by the IMU can be analysed by making use of some data points of the sample data.

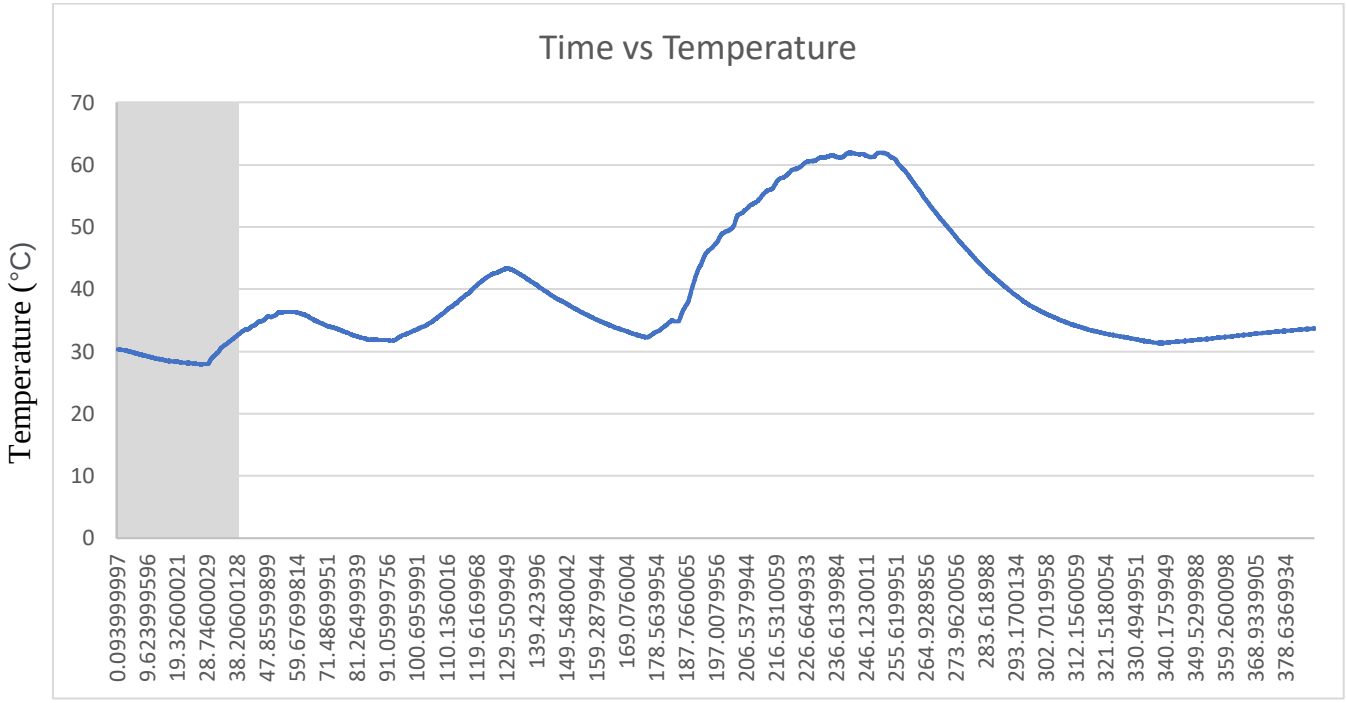

Figure 6 – Graph of time in seconds vs temperature readings

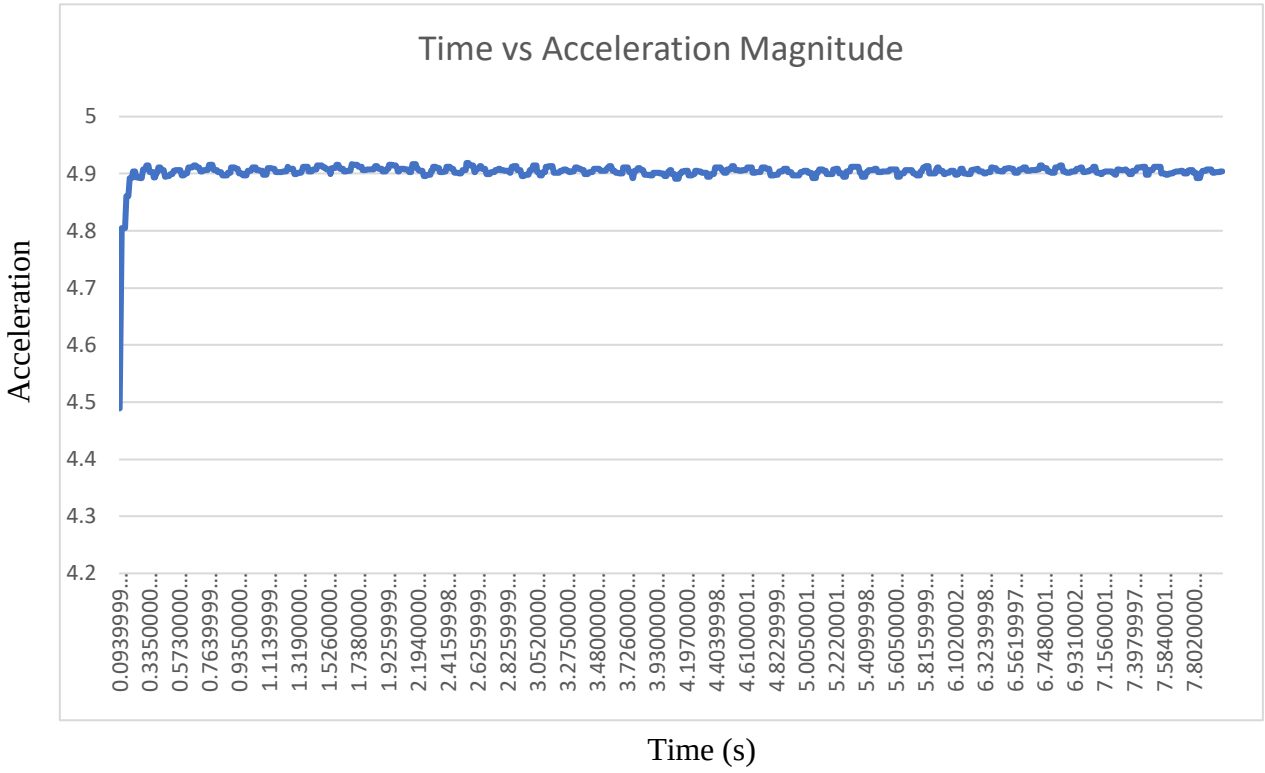

Figure 7 – Graph of time vs magnitude of acceleration

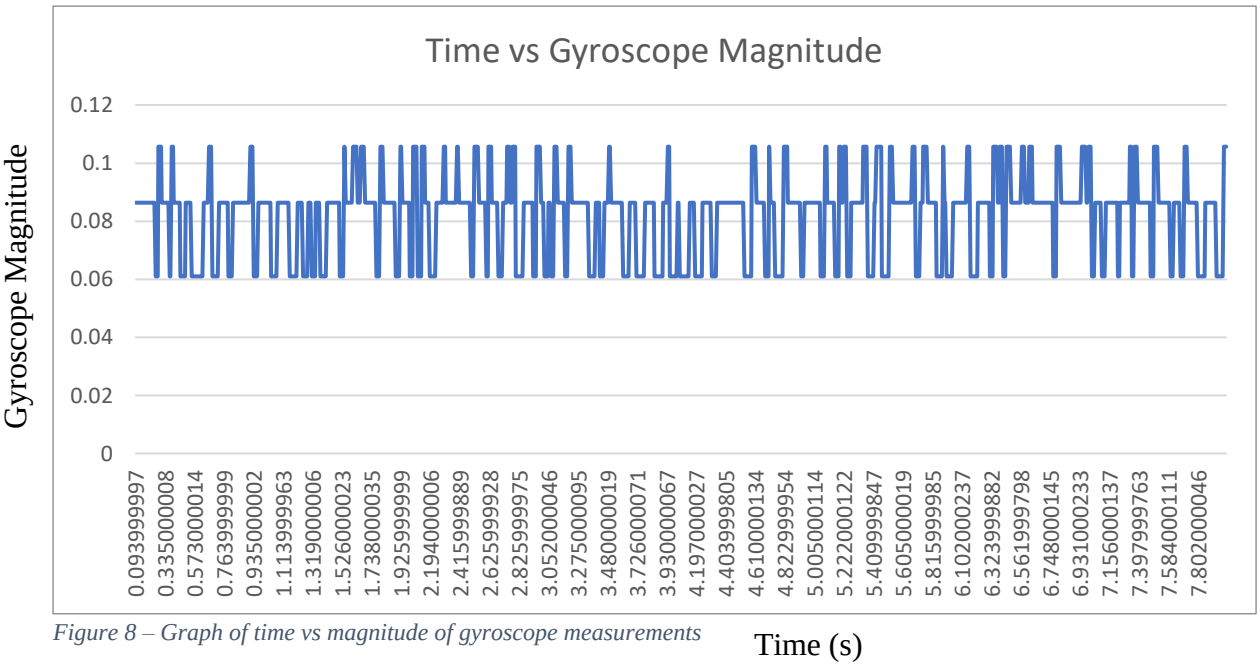

Figure 8 – Graph of time vs magnitude of gyroscope measurements

The following three graphs depict the Fourier transform of the simulated data. These graphs show what the data sets should produce once they have been decrypted, decompressed and

analysed using software such as Matlab. Matlab_R2022a was used to produce the following graphs.

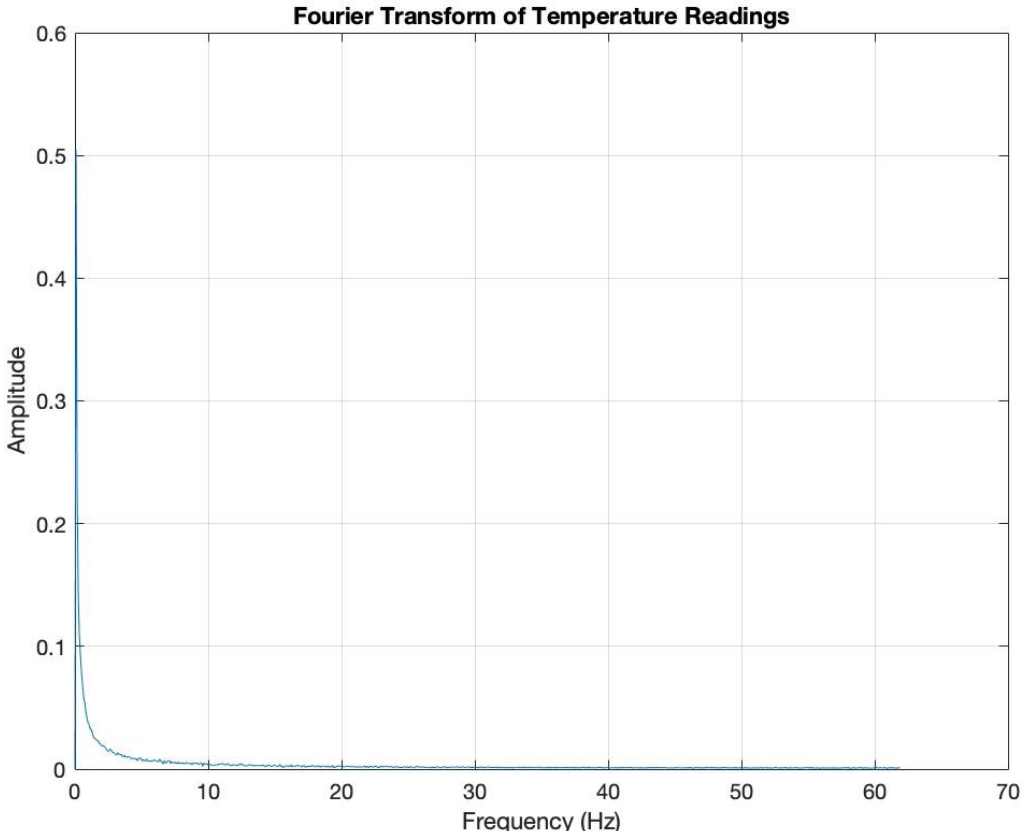

Figure 9 – Graph displaying the Fourier Transform of the temperature data

Since the Fourier graph is symmetric it was decided to only plot one side of the graph as it would be the same peak values for the negative frequencies. The graph displays a peak at the origin and then tends to towards zero as the frequency increases. The data set is more linear than random, since the temperature rarely fluctuates when compared to the graphs below.

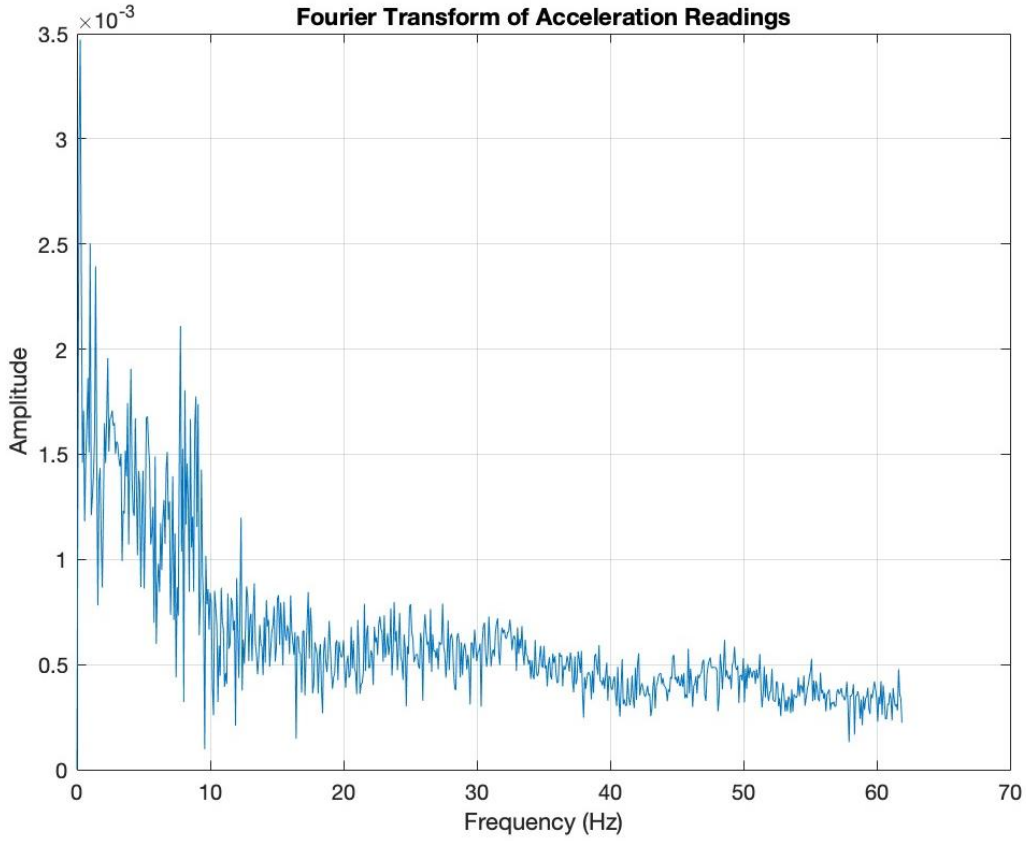

Figure 11 – Graph showing the Fourier Transform of the acceleration data

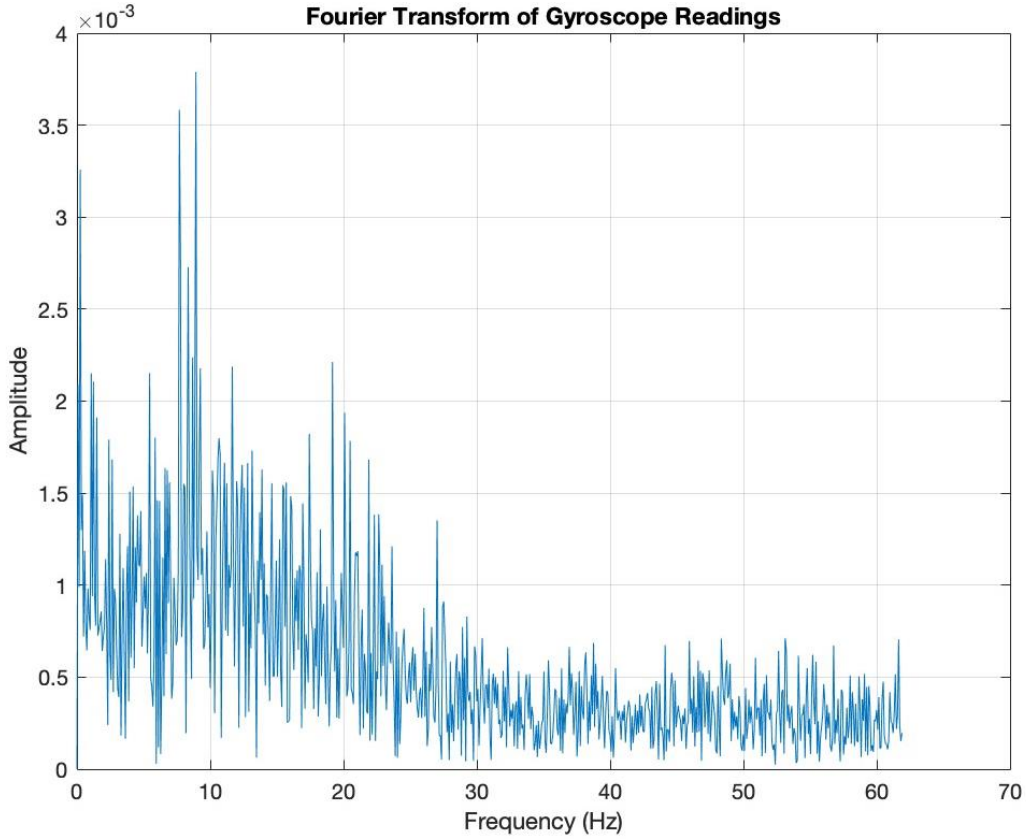

Figure 10 – Graph showing the Fourier Transform of the gyroscope data

As seen above, significant noise is present in the graphs except for temperature – which has a

spike at the origin, whereas the others follow an unexpected relationship. There is multiple fluctuations in the data, this indicates that the correlation coefficient between each data set is higher when compared to the data set of temperature. The above graphs represent the data in the frequency domain rather than the time domain, this is normally done for digital signal processing. The samples within the first 20ms were used to plot the Fourier transform.

2. Experiment Setup

2.1. Experiments to Check Overall Functionality of the System

The overall System uses a combination of encryption and compression functions in one Python file. The compression block will compress the input data file, passing the output file to the encryption block to encrypt the compressed file. The encrypted file can then be decrypted and decompressed to retrieve the original data. After decryption and decompression, a Python built-in comparison function is then used to make a deep comparison between the original data file and the resultant data file. Since both the compression and encryption algorithms are lossless, the input and output files should be identical. The purpose of these tests is to determine quantities associated to each feature of the system which can be used to verify the ATPs and determine whether the figures of merit have been met indicating a successful system or possible points for improvement. Particularly the figure of merits outlined in section 2 of part C above.

2.2. Experiments for Compression Block

The compression block was tested across various use cases to determine values that could be used as comparisons for respective ATPs determined in the paper design. All the tests were executed across the same three sets of sample data:

- "EEE3097S_2022_Turntable_Example_Data_2.csv"
- "EEE3097S_2022_Turntable_Example_Data.csv"
- "EEE3097S_2022_Walking_Around_Example_Data.csv"

While the tests have peripheral objectives related to the ATPs, the main aim of the compression block is to reduce the physical space required to store the data whilst retaining all data inside the file. The Lempel–Ziv–Markov chain algorithm (LZMA) was used for compression. The original file size is compared to the compressed file size for each sample data set to determine the compression ratio, this can be used to verify if a compression ratio of at least 1.5 is achieved. The execution time of the compression and decompression for each sample data set can also be determined using the Python time library. Lastly, the input and output files are compared for each sample data set to determine if the compression and decompression are lossless. This execution time can be used to determine if the compression block meets the figure of merit number 4 in section 2 of part C above.

2.3. Experiments for Encryption Block

Testing the encryption block consisted of various stages. Testing was done in phases to obtain values which could then be used for comparisons and verify the respective ATPs. All the tests were done on the sample data provided by the compression algorithm:

- "EEE3097S_2022_Turntable_Example_Data_2.csv.lz"
- "EEE3097S_2022_Turntable_Example_Data.csv.lz"
- "EEE3097S_2022_Walking_Around_Example_Data.csv.lz".

While the tests have peripheral objectives related to the ATPs, the primary function of the encryption block is to convert plaintext data into ciphertext, which cannot trivially be decrypted without the password. The encryption algorithm used was the Twofish algorithm developed by Bruce Schneier. The original file was compared to the encrypted file for each file to determine whether the encryption was successful. The execution time for encryption and decryption of each data file was determined using the built-in Python time library as means of verifying that the ATP has been met, specifically the execution time of the algorithm should not exceed 10 seconds per 10 kilobytes of data. Determining if the encryption was lossless was done by comparing the files provided to the algorithm (the compressed files) and the files following decryption; this was done to ensure the encryption is lossless as to verify the ATP. The execution time can be used to determine if the encyrption block meets the figure of merit number 3 in section 2 of part C above.

2.4. Expected Data to be Retrieved and Returned From Each Block

The compression block is expected to retrieve IMU data as a CSV file. The compression block will read the CSV file and output the compressed .lz file. The encryption block will then retrieve this .lz file and encrypts it into a .lz.tf file. The decryption algorithm will then retrieve the .lz.tf file and decrypt it into a .lz file. This .lz file is retrieved by the decompression algorithm, which will output a CSV file.

3. Results

3.1. Results of Experiments to Check Overall Functionality of the System

The output for the overall functionality experiment is shown below:

```
Uncompressed file size = 22429420 Bytes
Compressed file size = 1488504 Bytes
Compression took 42.5765600204 seconds

Enter encryption password: 1234567890
Unencrypted file size = 1488504 Bytes
Encrypted file size = 1488512 Bytes
Encryption took 5.5818529129 seconds

Enter encryption password: 1234567890
Encrypted file size = 1488512 Bytes
Unencrypted file size = 1488504 Bytes
Decryption took 5.675347805 seconds

Uncompressed file size = 1488504 Bytes
Compressed file size = 22429420 Bytes
Decompression took 2.6749780178 seconds

The original file matches the output file!
```

Figure 12 – Command-line output from the overall functionality experiment

The sample data set used for this experiment was "EEE3097S_2022_Turntable_Example_Data.csv". The uncompressed file has a size of 22429420 bytes and a compressed size of 1488504 bytes – this which means that a compression ratio of 15.1 was achieved. The compression took 42.58 seconds, and the decompression took 2.67 seconds. The encryption took 5.58 seconds, and the decryption took 5.68 seconds. The encryption added 8 bytes to the compressed file because the compressed file was not a multiple of 16 bytes (block size). These 8 bytes acted as padding for the encryption algorithm so that the file's content was correctly encrypted. The last line output in the figure above shows that the original file is identical to the decompressed file; therefore, the compression and encryption are both lossless.

3.2. Results of Experiments for Compressions Block

The output for the compression experiment is shown below in table form:

Table XIX – Raw results for the compression block experiments

File	Original File Size [bytes]	Compressed File Size [bytes]	Compression Time [seconds]	Decompression Time [seconds]
Turntable_1	22 429 420	1 488 504	41.6755	2.61922
Turntable_2	19 827 842	939 264	30.4814	1.7084
Walking	14 332 601	1 353 856	29.0786	2.7873

The values in the table below were calculated from those in the table above.

Table XX – Calculated results for the compression block experiments

File	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]
Turntable_1	15.0684	538 192.5385	8 563 403.6920
Turntable_2	21.1099776	650 489.8061	11 605 816.5600
Walking	10.58650329	492 891.4608	5 142 029.8360
Average	15.5883	560 624.6018	8 437 083.3630

As can be seen from the results shown in the table above, LZMA appears to be a high-speed and efficient compression algorithm. These results also clearly show that the decompression algorithm is significantly more efficient than the compression algorithm. This means that when the algorithms must be optimized for the STM more effort should be focused on making the compression more efficient rather than the decompression.

3.3. Results of Experiments for Encryption Block

The output for the encryption experiment is shown below in table form:

Table XXI – Raw results for the encryption block experiments

Compressed File	Original File Size [bytes]	Encrypted File Size [bytes]	Encryption Time [seconds]	Decryption Time [seconds]
Turntable_1.lz	1 488 504	1 488 512	5.5718	5.5043
Turntable_2.lz	939 264	939 264	2.2277	2.3134
Walking.lz	1 353 856	1 353 856	4.4566	4.6865

The values in the table below were calculated from those in the table above.

Table XXII – Calculated results for the encryption block experiments

File	Used Padding	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
Turntable_1	Yes	267 147.5059	270 425.3578
Turntable_2	No	421 626.4158	406 010.3635
Walking	No	303 787.9655	288 885.5455
Average	-	330 853.9624	321 773.7556

As seen by the data, we can conclude that Twofish is slightly faster at encrypting files than decrypting files. It is still, however, a high-speed and efficient algorithm. While the encryption and decryption speeds are both efficient, the implementation of the algorithms on the STM will impact the efficiency of the algorithms. This means that the algorithms will have to be optimized for the STM.

```
Encrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz:
Enter encryption password: 1234567890
Unencrypted file size = 1353856 Bytes
Encrypted file size = 1353856 Bytes
Encryption took 4.532520771 seconds

Decrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz.tf:
Enter encryption password: 1234567890
Encrypted file size = 1353856 Bytes
Unencrypted file size = 1353856 Bytes
Decryption took 4.4951100349 seconds

The original file matches the output file!
```

Figure 13 – Decryption using the correct password

```
Encrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz:
Enter encryption password: 1234567890
Unencrypted file size = 1353856 Bytes
Encrypted file size = 1353856 Bytes
Encryption took 4.514988184 seconds

Decrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz.tf:
Enter encryption password: t
Encrypted file size = 1353856 Bytes
Unencrypted file size = 1353856 Bytes
Decryption took 4.576939106 seconds

The original file does not match the output file!
```

Figure 14 – Decryption using an incorrect password

As can be seen by comparing Figure 10 and Figure 11, the decryption fails when an incorrect password is provided to the decryption algorithm.

3.4. Effects of Changing the Data Provided to the System

Noise was added to the data points in the "EEE3097S_2022_Turntable_Example_Data.csv" to determine whether such a change in the data will affect the compression performance or encryption performance. As seen in the figure below, the performance of both the compression and encryption has been severely affected compared to the result shown in Figure 9, which used the original data set.

```
Uncompressed file size = 22422023 Bytes
Compressed file size = 9091860 Bytes
Compression took 62.0316669941 seconds

Enter encryption password: 1234567890
Unencrypted file size = 9091860 Bytes
Encrypted file size = 9091872 Bytes
Encryption took 462.5292720795 seconds

Enter encryption password: 1234567890
Encrypted file size = 9091872 Bytes
Unencrypted file size = 9091860 Bytes
Decryption took 459.6317350864 seconds

Uncompressed file size = 9091860 Bytes
Compressed file size = 22422023 Bytes
Decompression took 14.3226840496 seconds

The original file matches the output file!
```

Figure 15 – Command-line output from the overall functionality experiment with noise

A table presenting the difference in the results between those in section 3.1 and this experiment is shown below:

Table XXIII – Changes due to adding noise to the data set

	Change in Input File Size [bytes]	Change in Output File Size [bytes]	Change in Speed [bytes/second]	Change in Speed (with original data set as baseline) [Percentage]
Compression	4397	7 603 356	165 342.3952	-31.4%
Decompression	7 603 356	4397	6 819 408.5070	-81.3%
Encryption	7 603 356	7 603 360	247 011.6069	-92.6%
Decryption	7 603 360	7 603 356	242 496.0141	-92.5%

As can be seen, by the severe changes in performance, adding noise to the data set will harm the performance of both compression and encryption.

4. Simulated Data Acceptance Test Procedures

4.1. Compression ATPs

Table XXIV – Simulation Data ATPs for the compression block

ATP	Description	ATP achieved	Comment
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data.	Yes	Table III shows that the compression algorithm achieved an average speed of ~560 000 bytes per second, which is far more than 2048 bytes per second.
Data loss	No data must be lost during the compression of the data file.	Yes	LZMA uses lossless compression, validated by the deep file comparison in section 3.1.
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5.	Yes	Table III shows that the compression algorithm achieved an average compression ratio of ~15.6, which is far more significant than 1.5.

4.2. Encryption ATPs

Table XXV – Simulated Data ATPs for the encryption block

ATP	Description	ATP achieved	Comment
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	Yes	Table V shows that the encryption algorithm achieved an average speed of ~330 000 bytes per second, surpassing the ATP.
Data loss	No data must be lost during the encryption and decryption of the data file.	Yes	As shown in section 3.1, the compression and encryption are both lossless.
Encryption security	The encryption must be strong enough to prevent trivial decryption.	Yes	This ATP is difficult to test, but the encryption prevents trivial decryption attempts, as shown in section 3.3, figures 10 and 11.
Encryption integrity	The original data in the file must be identical to the decrypted data in the output file.	Yes	As shown in section 3.1, the compression and encryption are both lossless.

4.3. Checksum ATPs

Table XXVI – Simulated Data ATPs for the checksum block

ATP	Description	ATP achieved	Comment
STM checksum	The STM must create a checksum of the original data file successfully.	No	No data collection from the STM could be created; therefore, no checksum.
Client checksum	The client's device must successfully create a checksum of the decrypted and decompressed file.	Yes	The checksum for the client was created using Python successfully.
Checksum comparison	The client's and STM's checksums must be identical.	No	Unable to create checksum on STM.

F. Validation using the IMU

1. IMU Module

1.1. Additional Features

The system used the ICM-20948 MEMS motion tracking unit provided on the SparkFun 9DoF IMU Breakout. The ICM-20948 is the same sensor provided by the Sense Hat B. The decision was made to use only the MEMS motion tracking unit as it provides the system with only the needed data and requires less setup and configuration to communicate with the STM microcontroller. The Sense Hat B was provided to the team after successfully communicating with the ICM-20948 on the SparkFun; therefore, it was decided that we would not be using the Sense Hat B, although the system can be configured to use either sensor.

The most noticeable feature of using the ICM-20948 was that it consumes significantly less power when compared to the Sense Hat B, it consumes 2.5mW when all 9-axis sensors are implemented, since the Sense Hat B has unused additional features which increase its power consumption. The ICM-20948 has the following specifications as outlined by the data sheet:

- 3-Axis Gyroscope with Programmable FSR of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps
- 3-Axis Accelerometer with Programmable FSR of ±2g, ±4g, ±8g, and ±16g
- 3-Axis Compass with a wide range to ±4900 μT
- Auxiliary I2 C interface for external sensors
- On-Chip 16-bit ADCs and Programmable Filters
- 7 MHz SPI or 400 kHz Fast Mode I²C
- Digital-output temperature sensor
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

Since there was no utilisation of all the features of the Sense Hat B since it did not replicate what will be implemented on the SHARC buoy, the SparkFun 9DoF IMU Breakout would be the only sensor used for testing the system. The ICM-20948 will provide recordings similar to what will be used on the SHARC buoy, which is the ICM-20649, as well as containing the same features (outlined above) as the Sense Hat B and more features than the ICM-20649. However the ICM-20948 and the ICM-20649 have the same features above with the exception of the compass being present on the ICM-20948. The 400kHz Fast Mode I^2C is beneficial for this project as it allows for fast testing and communication, this allows for the quick identification of errors as well uploading new code to the STM.

1.2. Testing of the IMU ensuring the system can be extrapolated to the buoy:

The ICM-20948 used for testing the entire system is made by the same company as the ICM-20649 onboard the SHARC buoy. It was decided to perform all tests solely with the SparkFun IMU instead of using the Sense Hat B. The SparkFun IMU is a 9-axis sensor, whereas the ICM-20649 is a 6-axis sensor. Although the experiments in the following sections used all nine axes provided by the SparkFun IMU, these same results can be extrapolated for using only the six axes provided by the ICM-20649.

The ICM-20649 only records the gyroscopic and accelerometer data, whereas the SparkFun IMU records gyroscopic, accelerometer, and magnetic data. There are no other significant differences between the SparkFun IMU and the one on the SHARC buoy. The SparkFun IMU provided more data than necessary, and this was done to ensure that the system could be extrapolated effectively to work on the SHARC buoy. The climate differences could not be tested since we cannot reproduce the conditions of Antarctica in South Africa.

1.3. Validation test for the IMU module:

Tests were done on the IMU to ensure that the STM was successfully powered on. The connection between the IMU and the STM was tested using the computer's serial interface to ensure the code was run successfully on the STM. The IMU's data was then collected and translated from binary to decimal to ensure that the readings were accurate compared to the ambient surroundings. The IMU was then moved around and placed in different environments to test if the data set would change, indicating that the sensor was working as expected. This data was collected and tabulated for comparisons to be made and to verify the success of the entire system.

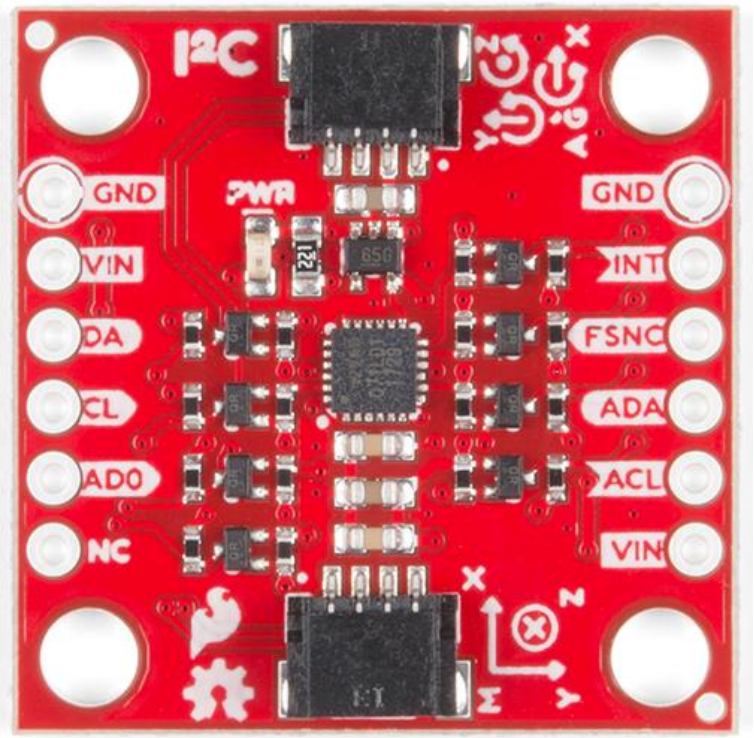

Figure 16 – The SparkFun 9DoF IMU Breakout

The table below shows the IMU first sitting flat on a table. The IMU was then placed so that its pins were perpendicular to the table, hence measuring gravity's downwards force on the sensor's x-axis. The IMU was then placed so that the pins were parallel to the table, hence measuring gravity's downwards force on the sensor's negative y-axis.

Table XXVII – Raw sensor measurements from the accelerometer

Reading Group	Time	Accel X	Accel Y	Accel Z
Control	0:00:00.845806	276	40	16451
	0:00:01.951676	278	43	16455
	0:00:03.057521	281	45	16460
	⋮	⋮	⋮	⋮
Perpendicular to table	0:00:11.901786	11842	73	5424
	0:00:13.007536	13847	68	3345
	0:00:14.112257	15892	73	1250
	⋮	⋮	⋮	⋮
Parallel to table	0:00:36.151303	247	-4492	13530
	0:00:37.257039	222	-6540	11816
	0:00:38.427122	211	-9918	9328

The table below shows the IMU first sitting flat on a table. The IMU was then placed so that its pins were almost perpendicular to the table; hence the gyroscope's y-axis changed. The IMU was then placed so that the pins were parallel to the table; hence the gyroscope's x-axis changed.

Table XXVIII – Raw sensor measurements from the gyroscope

Reading Group	Time	Gyro X	Gyro Y	Gyro Z
Control	0:00:00.325570	2	0	-1
	0:00:01.429950	2	1	0
	0:00:02.534451	1	1	0
	⋮	⋮	⋮	⋮
Perpendicular to table	0:00:04.744082	5	-148	32
	0:00:05.848769	4	-151	32
	0:00:06.953420	4	-144	32
	⋮	⋮	⋮	⋮
Parallel to table	0:00:25.739159	-147	-36	81
	0:00:26.844462	-161	58	80
	0:00:27.950040	-173	31	79

The table below shows the IMU first sitting flat on a table without a strong magnet in its vicinity. A magnet was placed directly above the sensor (parallel with the sensor's z-axis). The magnet was then placed parallel to the sensor's pins (parallel with the sensor's x-axis). Lastly, the magnet was placed perpendicular to the sensor's pins (parallel with the sensor's y-axis).

Table XXIX – Raw sensor measurements from the magnetometer

Reading Group	Time	Magn X	Magn Y	Magn Z
Control	0:00:00.647816	-956	-119	44
	⋮	⋮	⋮	⋮
Magnet on z-axis	0:00:10.587180	-2338	-588	7831
	0:00:11.691698	-2021	-1962	10364
	0:00:12.796130	-1763	-3298	12830
	⋮	⋮	⋮	⋮
Magnet on x-axis	0:00:33.780674	-1683	-118	451
	0:00:34.885170	-2690	-182	717
	0:00:35.989632	-3695	-267	888
	⋮	⋮	⋮	⋮
Magnet on y-axis	0:00:54.765832	-1056	-446	137
	0:00:55.870350	-1151	-1164	290
	0:00:56.975013	-1243	-2174	556

2. Experiment Setup

2.1. Experiments to Check Overall Functionality of the System

The overall system uses a combination of encryption and compression functions compiled in C and managed by a python script for execution and testing. A checksum is performed on the collected sensor data from the IMU (stored on the STM microcontroller temporarily), the compression block will compress the data, and encryption is performed on the compressed data before being transmitted to the computer for testing. The encrypted data can then be decrypted and decompressed to retrieve the original data by passing the encrypted data back to a secondary STM, which has the decryption and decompression algorithm loaded; the STM will then perform the subsequent algorithms before returning the original data along with a checksum (which is generated on the secondary STM) for confirmation of the retrieval and storage of data to be confirmed – the checksum is used to ensure that the original sensor data is identical to the resultant data after decompression. Since both the compression and encryption algorithms are lossless, the input and output files should be identical. The checksum confirms this – this ensures that the efficiency of the overall setup is 100%.

A timer was used to determine the time taken for the system to collect a block of sensor data and perform the necessary algorithms on the data before transmission; this test was also performed on the decompression and decryption to ensure the data processing on the STM satisfies the requirements. The total time for the overall system to perform the above actions were used to verify the requirements that the encryption and compression blocks were efficient and did not exceed the time constraints outlined in the figure of merits. Once decompression and decryption had been performed, the original size of the sensor data was compared to the compressed data size to ensure that a compression ratio of at least 1.5 was achieved every iteration.

2.2. Experiments for Compression Block

Testing was done in phases to obtain values which could then be used for comparisons and verify the respective ATPs along with determining figures of merit. The compression algorithm was tested by performing compression on multiple sets of known data generated by a pseudorandom value generator using a seed. The seed used on the STM and the computer were the same – hence the data compressed on the STM and the data compressed on the computer were the same. This was done to validate that the algorithm is working on the STM as it was on the computer – confirming the algorithm is working as desired before the sensor data is passed to the compression block. Since the computer runs the same algorithm as the STM, the output file size and the output data must be the same as the ones returned by the computer. While the tests have peripheral objectives related to the ATPs, the main aim of the compression block is to reduce the physical space required to store the data whilst retaining all data inside the file. It must be emphasised that the tests focused predominantly on the sensor data rather than the seed data.

The Lempel-Ziv-Storer-Szymanski (LZSS) compression algorithm was used for compression. The original sensor data set size is compared to the compressed data size for each iteration to determine the compression ratio – the compression ratio should be at least 1.5. The execution time of the compression and decompression for each generated data set can be determined via the same methods used for the overall system; however, in this section, the file size was divided by the time taken to compression to ensure that at least one kilobyte of data is compressed every second.

Lastly, the input and output data sets are compared for each sample data set to determine if the compression and decompression are lossless – the checksum is also used to verify that the data collected by the sensor as well as the data generated by the STM using the seed is the same as the data in the output file following decompression. The process described above was repeated using data collected by the IMU and stored on the STM; the results from the generated data set were used as a control to verify that the compression block performed as expected.

2.3. Experiments for Encryption Block

Testing the encryption block consisted of various stages. Testing was done in phases to obtain values which could then be used for comparisons and verify the respective ATPs. The encryption algorithm was tested by performing encryption on multiple sets of known data generated by a pseudorandom value generator using a seed. Since the computer used for testing is a 64-bit processor while the STM microcontroller has a 32-bit processor, the machines' processors perform substitutions of the data into the S-boxes differently. This raised the issue that the computer cannot be used to verify the algorithm. To solve this problem, the sensor data encrypted by the STM is transferred to the client's computer along with the padding length used for the encryption and the checksum of the sensor data generated by the STM. The client can then see all the information related to the encryption before sending the data to another STM for decryption.

The generated data was used to confirm that the algorithm is working as desired before the use of real-world sensor data– this was done as the generated data remained constant and was easily repeatable. The time taken for the computer to perform encryption on the generated data was recorded using a timer. This execution time is used to verify that the encryption block works as desired. The above process once validated with the generated data, the entire testing procedure was repeated using sensor data collected by the STM. This was done to show that the encryption block operates the same with the real world data as it did with known generated data sets, thus verifying all ATPs and figures of merit and concluding that the encryption block works as desired.

Since the STM generated a checksum when the data was collected, this checksum can be used for comparison against the checksum generated following decryption to ensure that the encryption is lossless and that the client receives all data. While these tests have peripheral objectives related to the ATPs, the primary function of the encryption block is to convert plaintext data into ciphertext, which cannot trivially be decrypted without the password. The encryption algorithm used was a custom modified blowfish algorithm which Bruce Schneier originally developed – Twofish was unable to be implemented on the hardware of the STM [4]. Hence, we were forced to revert to Twofish's predecessor, which was designed for hardware use. The original data was compared to the encrypted data for each data set to

determine whether the encryption was successful. The execution time for encryption and decryption of each data set was determined using a timer. Determining if the encryption was lossless was done by comparing the raw sensor data provided to the encryption algorithm and the output data following decryption; the checksum was also used to verify as a secondary measure. The above information was then used to confirm that the figure of merits were satisfied to meet the user requirements. The encryption algorithm must be able to encrypt at least one kilobyte of data per second. As outlined above, the testing procedure for the compression block is similar to that of the encryption block by using seed generated data and sensor data, this is done to reliably know that the system passes all figures of merit without doubt. It must be emphasised that the tests focused predominantly on the sensor data rather than the seed data.

2.4. Experiments for Checksum Block

Testing the checksum block can be done by providing the CRC32 checksum algorithm with identical and different data sets and creating a checksum of each. To verify the CRC32 algorithm, a checksum of the sensor data collected by the remote STM is generated and transmitted with sensor data. Then the client's device receives the checksum and sensor data and sends it to their local STM. If the resultant 32-bit checksums of each data set are identical, then the data set has not changed. However, if the checksums are different, then the data is different, or in a rare case, the checksum was corrupted during transmission.

2.5. Expected Data to be Retrieved and Returned From Each Block

The compression block is expected to receive the raw IMU data in binary at a rate of 1 sample per 500 milliseconds, which will be compressed by executing the compression algorithm on the collected sensor data. The compressed data will also be binary and stored on the STM. The compressed binary data will be passed to the encryption block on the STM. The encryption block will then execute the encryption algorithm to convert the compressed sensor data into ciphertext before transmission to the client's computer. The client will then send the encrypted, compressed data in ciphertext to their STM for decryption. The decryption block will then execute its algorithm on the data which the client supplies. Following this, the decryption block will pass the decrypted data (compressed data) in binary to the decompression block, which will decompress the data so that the client can read the IMU's sensor data.

3. Results

3.1. Results of Experiments to Check Overall Functionality of the System

The figures below show the command-line output for a single overall functionality test of the system. This test was conducted with 132 bytes of collected sensor data. This data was then passed to the compression block. The compressed data was then passed to the encryption block, and the encrypted data was transmitted to the user. The above process is shown in Figure 4 below. The user is then asked if they want to decrypt and decompress the data. The outputs from the encryption block and compression block is shown in Figure 4 and 5 below.

```
SENSOR DATA:
Sensor data size: 132 bytes
0   ff ff ff ff 00 25 00 03  08 07 00 03 00 02 00 00
1   ff 91 00 07 00 0b ff ff  ff ff 00 48 00 08 10 0d
2   00 02 00 02 00 00 ff 24  00 0e 00 15 ff ff ff ff
3   00 6d 00 10 18 11 00 02  00 01 00 00 fe b7 00 15
4   00 20 ff ff ff ff 00 90  00 16 20 17 00 02 00 01
5   ff ff fe 4b 00 1a 00 2b  ff ff ff ff 00 b4 00 1b
6   28 1b 00 02 00 01 ff ff  fd dd 00 21 00 35 ff ff
7   ff ff 00 d8 00 21 30 1f  00 01 00 01 ff ff fd 70
8   00 28 00 40
CRC32 CHECKSUM:
0   b9 81 7e 70
COMPRESSED DATA:
Compression took: 189.0 milliseconds
Compressed data size: 103 bytes
0   ff 80 0a 01 25 80 40 e1  10 70 18 c0 20 40 f2 c8
1   c0 01 23 0b 0a a5 22 01  08 88 43 42 64 15 39 24
2   02 1d 00 8a 85 52 db 00  88 46 22 21 32 80 85 47
3   fd b7 09 8c 02 40 15 4c  84 02 2d 20 8b 85 4c 16
4   3f ea 5c 02 35 00 95 85  53 69 00 8d ca 23 61 55
5   fe f7 60 12 18 04 d4 2a  9d 80 48 cc 23 e3 f2 0a
6   9d c2 01 28 80 50 00
ENCRYPTED DATA:
Encryption took: 118.489 milliseconds
Encrypted data size: 104 bytes
0   de fa d8 d4 ad 4b 97 1a  39 30 c9 f1 8f 14 f3 4c
1   f4 a6 01 37 a4 8c 1f 9a  84 ad 35 64 b6 c0 ed 07
2   d4 be 12 c0 c0 55 ac 02  2b cf 8f 05 a4 6b 14 22
3   ec 60 a1 ae 15 dc 63 b0  b4 d8 88 f7 40 35 29 2d
4   0d f1 f2 42 61 90 bd 4e  40 c6 97 2c 7e 75 23 36
5   43 0f 09 7e e4 84 21 6d  a3 ce f7 5a c7 a2 56 76
6   ca 45 95 ee f0 e3 62 83
COMPRESSED DATA SIZE
0   67 00 00 00
PADDING LENGTH:
0   01
```

Figure 18 – STM1 CLI output for the overall functionality experiment

```
Decrypt and decompress? (y/n): y
PADDING LENGTH CONFIRMATION:
0   01
COMPRESSED DATA SIZE CONFIRMATION:
0   67 00 00 00
ENCRYPTED DATA CONFIRMATION:
0   de fa d8 d4 ad 4b 97 1a  39 30 c9 f1 8f 14 f3 4c
1   f4 a6 01 37 a4 8c 1f 9a  84 ad 35 64 b6 c0 ed 07
2   d4 be 12 c0 c0 55 ac 02  2b cf 8f 05 a4 6b 14 22
3   ec 60 a1 ae 15 dc 63 b0  b4 d8 88 f7 40 35 29 2d
4   0d f1 f2 42 61 90 bd 4e  40 c6 97 2c 7e 75 23 36
5   43 0f 09 7e e4 84 21 6d  a3 ce f7 5a c7 a2 56 76
6   ca 45 95 ee f0 e3 62 83
DECRYPTED DATA:
Decryption took: 117.606 milliseconds
Decrypted data size: 103 bytes
0   ff 80 0a 01 25 80 40 e1  10 70 18 c0 20 40 f2 c8
1   c0 01 23 0b 0a a5 22 01  08 88 43 42 64 15 39 24
2   02 1d 00 8a 85 52 db 00  88 46 22 21 32 80 85 47
3   fd b7 09 8c 02 40 15 4c  84 02 2d 20 8b 85 4c 16
4   3f ea 5c 02 35 00 95 85  53 69 00 8d ca 23 61 55
5   fe f7 60 12 18 04 d4 2a  9d 80 48 cc 23 e3 f2 0a
6   9d c2 01 28 80 50 00
DECOMPRESSED DATA:
Decompression took: 155.795 milliseconds
Decompressed data size: 132 bytes
0   ff ff ff ff 00 25 00 03  08 07 00 03 00 02 00 00
1   ff 91 00 07 00 0b ff ff  ff ff 00 48 00 08 10 0d
2   00 02 00 02 00 00 ff 24  00 0e 00 15 ff ff ff ff
3   00 6d 00 10 18 11 00 02  00 01 00 00 fe b7 00 15
4   00 20 ff ff ff ff 00 90  00 16 20 17 00 02 00 01
5   ff ff fe 4b 00 1a 00 2b  ff ff ff ff 00 b4 00 1b
6   28 1b 00 02 00 01 ff ff  fd dd 00 21 00 35 ff ff
7   ff ff 00 d8 00 21 30 1f  00 01 00 01 ff ff fd 70
8   00 28 00 40
CRC32 CHECKSUM:
Average checksum generation time: 0.067 milliseconds
0   b9 81 7e 70
```

Figure 17 – STM2 CLI output for the overall functionality experiment

The values in the table below were calculated from those in Table LIII in Appendix I.

Table XXX – Results for the overall functionality experiment

Round	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
1.1	1.294	665.008	807.883	727.433	831.897
1.2	1.245	640.267	804.388	689.078	831.391
1.3	1.245	655.074	770.443	720.387	805.964
2.1	1.320	728.139	827.353	695.717	860.08
2.2	1.361	774.275	844.892	927.553	863.651
2.3	1.375	803.849	852.207	859.268	836.769
3.1	1.424	857.358	863.486	894.743	868.056
3.2	1.385	820.664	849.636	869.187	880.906
3.3	1.424	849.053	864.459	906.367	886.798
4.1	1.397	845.544	865.350	888.901	889.012
4.2	1.500	897.126	866.628	874.943	892.07
4.3	1.571	951.588	872.595	863.740	875.625
5.1	1.650	1012.295	875.206	859.029	882.418
5.2	1.594	994.329	878.370	872.611	897.054
5.3	1.557	944.417	873.552	893.988	890.876
6.1	1.752	1083.560	882.278	895.642	895.933
6.2	1.744	1108.067	879.437	891.838	899.772
6.3	1.808	1122.544	883.668	892.807	884.973
7.1	1.798	1159.310	879.921	914.283	889.902
7.2	1.833	1161.148	884.703	909.643	889.828
7.3	1.763	1106.489	883.994	910.722	904.193
8.1	1.906	1208.282	885.591	911.553	901.437
8.2	1.963	1243.456	890.908	911.941	901.489
8.3	1.985	1272.341	890.942	919.509	899.435
Average	1.579	954.341	861.579	862.537	877.480

The table below shows the averages for each of the rounds recorded in Table LIII in Appendix I.

Table XXXI – Average results for the overall functionality experiment

Round	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
1	1.262	653.449	794.238	823.084	712.299
2	1.352	768.754	841.484	853.500	827.513
3	1.411	842.358	859.194	878.587	890.099
4	1.489	898.086	868.191	885.569	875.861
5	1.600	983.680	875.709	890.116	875.209
6	1.768	1104.724	881.795	893.559	893.429
7	1.798	1142.316	882.873	894.641	911.549
8	1.951	1241.360	889.147	900.787	914.334

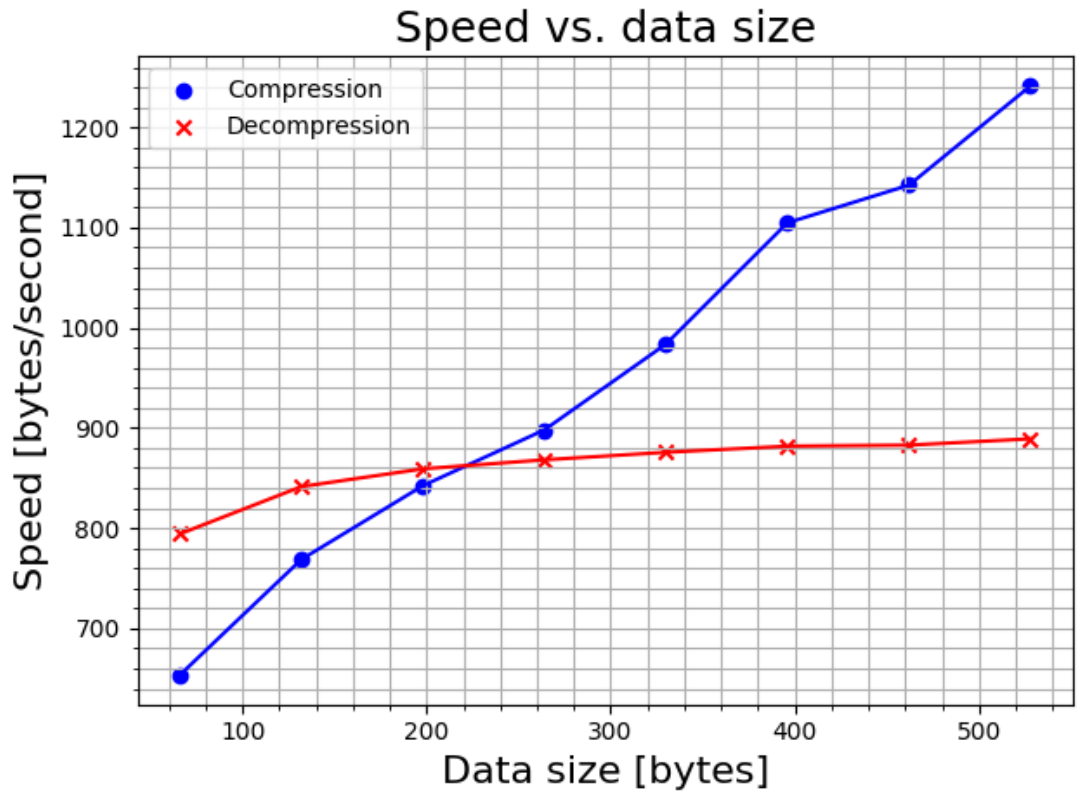

Figure 19 – Compression block speed vs input size for overall functionality experiment

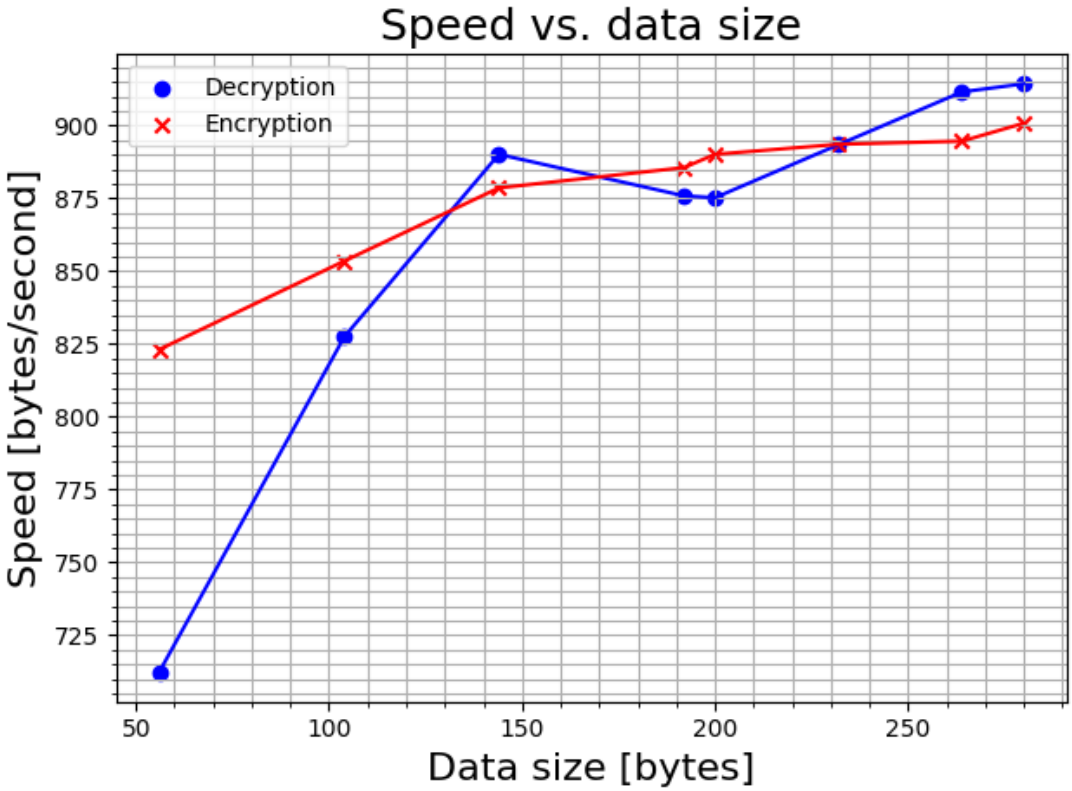

Figure 20 – Encryption block speed vs input size for overall functionality experiment

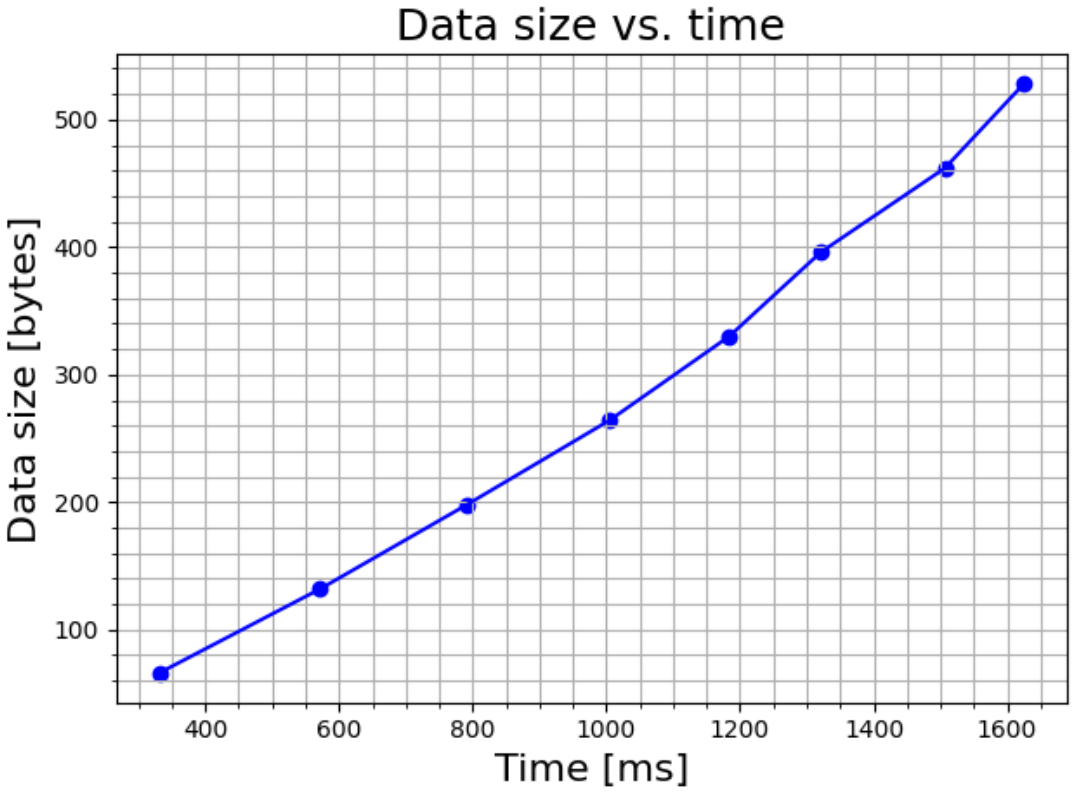

Figure 21 – Data size vs total computational time for overall functionality experiment

3.2. Results of Experiments for Compression Block

The values in the table below were calculated from those in Table LIV in Appendix II.

Table XXXII – Calculated results for the compression block experiments

Round	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]
1.1	1.269	658.801	817.186
1.2	1.245	654.944	797.516
1.3	1.200	652.329	806.166
2.1	1.304	760.187	845.934
2.2	1.231	743.014	850.595
2.3	1.313	771.257	835.093
3.1	1.546	940.434	872.979
3.2	1.554	927.021	866.672
3.3	1.702	1024.187	878.978
4.1	1.729	1082.594	884.685
4.2	1.800	1134.745	882.502
4.3	1.692	1058.931	877.195
5.1	1.954	1240.254	889.767
5.2	1.895	1189.047	884.336
5.3	1.909	1218.528	885.485
6.1	1.913	1274.27	892.212
6.2	2.026	1314.271	889.294
6.3	2.007	1302.769	890.199
7.1	2.034	1338.915	891.103
7.2	2.161	1404.943	896.095
7.3	2.011	1323.125	892.74
8.1	2.010	1313.297	896.205
8.2	2.080	1350.629	895.38
8.3	1.990	1328.593	895.107
9.1	2.190	1451.194	898.841
9.2	2.013	1340.257	896.556
9.3	2.079	1392.739	897.063
10.1	2.249	1497.538	901.686
10.2	2.229	1482.645	900.714

10.3	2.012	1324.798	896.501
Average	1.812	1149.875	876.826

The table below shows the averages for each of the rounds recorded in Table LIV in Appendix II.

Table XXXIII – Average results for the compression block experiment

Round	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]
1	1.238	655.358	806.956
2	1.283	758.153	843.874
3	1.601	963.881	872.876
4	1.741	1092.090	881.461
5	1.919	1215.943	886.529
6	1.982	1297.103	890.568
7	2.068	1355.661	893.312
8	2.027	1330.840	895.564
9	2.094	1394.730	897.487
10	2.163	1434.994	899.634

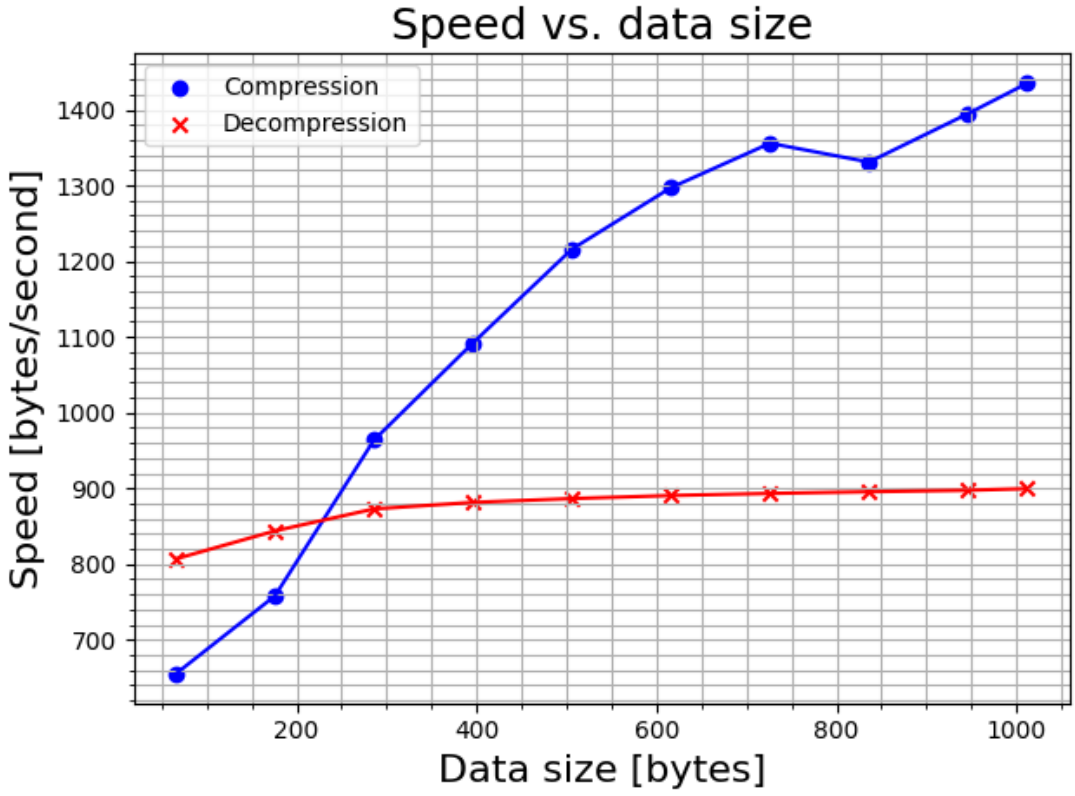

Figure 22 – Compression block computational speed versus input data size

The graph above indicates that the compression algorithm is more efficient than the decompression algorithm because as the size of the input data increases, the computational speed of the compression algorithm increases, reducing the system's total run time per block of sensor data. Whereas the decompression speed remains essentially constant, increasing the system's total run time per block of sensor data.

3.3. Results of Experiments for Encryption Block

The values in the table below were calculated from those in Table LV in Appendix III.

Table XXXIV – Calculated results for the encryption block experiments

Round	Padding Length [bytes]	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
1.1	6	890.065	845.527
1.2	6	915.099	840.346
1.3	6	887.749	839.317
2.1	4	887.230	888.343
2.2	4	901.056	888.372
2.3	4	909.194	878.422
3.1	2	909.848	881.803
3.2	2	900.580	892.961
3.3	2	905.814	895.283
4.1	0	900.016	892.848
4.2	0	896.782	902.672
4.3	0	896.916	899.814
5.1	6	923.521	904.785
5.2	6	923.125	906.956
5.3	6	913.801	900.247
6.1	4	915.181	905.879
6.2	4	918.993	904.178
6.3	4	918.240	906.696
7.1	2	910.489	910.354
7.2	2	913.016	909.743
7.3	2	913.605	907.322
8.1	0	912.585	908.789
8.2	0	913.119	911.528
8.3	0	909.700	906.428

9.1	6	920.522	907.335
9.2	6	921.796	911.689
9.3	6	917.932	908.150
10.1	4	921.302	911.882
10.2	4	918.424	910.872
10.3	4	920.881	913.040
Average	-	910.219	896.386

The table below shows the averages for each of the rounds recorded in Table LV in Appendix III.

Table XXXV – Average results for the encryption block experiment

Round	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
1	897.638	841.730
2	899.160	885.046
3	905.414	890.016
4	897.905	898.445
5	920.149	903.996
6	917.471	905.584
7	912.370	909.140
8	911.801	908.915
9	920.084	909.058
10	920.202	911.931

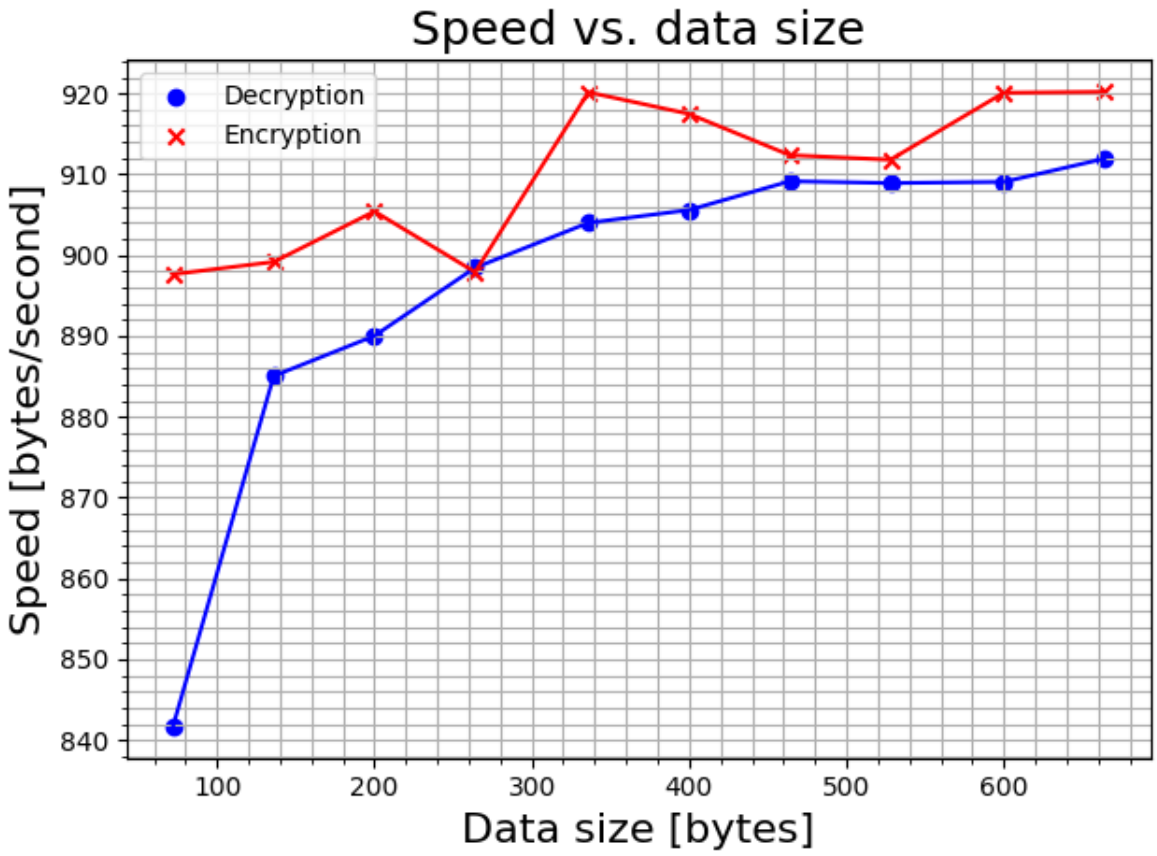

Figure 23 – Encryption block computational speed versus input data size

The graph above implies that as the size of the supplied data increases, the decryption and encryption speeds tend towards each other at about 915 bytes per second.

3.4. Results of Experiments for Checksum Block

The results for the checksum block experiments are shown below in table form.

Table XXXVI – Raw results for the checksum block experiments

Round	Data size	STM 1 Checksum	STM 2 Checksum	Average generation time [ms]
1	66	ea 54 a9 d5	ea 54 a9 d5	0.015
2	550	e5 a3 0b 4a	e5 a3 0b 4a	0.017
3	1034	bd 09 cb ed	bd 09 cb ed	0.019
4	1518	93 1b bb fe	93 1b bb fe	0.022
5	2002	f6 18 aa 7f	f6 18 aa 7f	0.024
6	2486	54 9d 79 3e	54 9d 79 3e	0.025
7	2970	5b f5 85 87	5b f5 85 87	0.029

8	3454	df 1b ae a0	df 1b ae a0	0.030
9	3938	0e 81 35 20	0e 81 35 20	0.033
10	4422	8c 39 d4 b7	8c 39 d4 b7	0.035

The checksums for every round match; therefore, the original sensor data matches the data received by the second STM. The values in the table below were calculated from those in the table above.

Table XXXVII – Average checksum generation speed for checksum algorithm

Round	Average generation speed [bytes/second]
1	4 400 000
2	32 400 000
3	54 400 000
4	69 000 000
5	83 400 000
6	99 400 000
7	102 000 000
8	115 000 000
9	119 000 000
10	126 000 000

To demonstrate the effectiveness of the checksum algorithm, the table below shows the effect when the data is changed, causing STM2 to generate a different checksum.

Table XXXVIII – Raw results for the checksum block experiments

Data size	STM 1 Checksum	STM 2 Checksum
66	74 51 92 4d	96 d1 38 a7
2002	fd 63 fa be	73 25 6a b6
4422	f3 fa 5f 39	a7 30 0d 27

Only the first byte of each data set was changed after the sensor data was transmitted to the second STM before checksum generation, and even due to such a minuscule change, the checksums do not match. This shows how the checksum can be used reliably to indicate whether the algorithms are lossless and that no data is lost during transmission between the STMs and the client. However, if, during transmission, the checksum is affected (i.e., a single bit flips), then the checksum would indicate that the transmitted data has changed even though it may still be unchanged.

3.5. Effects of Changing the Data Provided to the System

3.5.1. Under Sampling Experiment

The CLI output of under sampling the data provided to the system is shown below.

```
SENSOR DATA:
Sensor data size: 132 bytes
0   ff ff ff ff 00 22 00 04  08 08 00 01 00 01 ff ff
1   ff 92 00 07 00 0b ff ff  ff ff 00 46 00 09 10 0e
2   00 01 00 01 ff ff ff 25  00 0e 00 16 ff ff ff ff
3   00 6a 00 0f 18 15 00 02  00 01 ff ff fe b8 00 16
4   00 21 ff ff ff ff 00 8b  00 14 20 1d 00 02 00 02
5   ff ff fe 4a 00 1e 00 2b  00 00 00 00 00 00 00 00
6   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
7   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8   00 00 00 00
CRC32 CHECKSUM:
0   d4 7e 8f e6
COMPRESSED DATA:
Compression took: 139.111 milliseconds
Compressed data size: 74 bytes
0   ff 80 0a 01 22 80 41 21  10 88 04 04 02 20 c2 c9
1   40 20 f0 08 58 55 28 d0  08 4c 42 1c 15 69 2c 00
2   12 31 60 aa 5a a0 10 f8  c4 56 01 02 0a 9f fb 70
3   13 18 04 84 2a 98 b8 04  52 41 1d 0a 94 08 2a 54
4   a8 04 7a 01 2b 33 f8 03  c0 16
ENCRYPTED DATA:
Encryption took: 93.338 milliseconds
Encrypted data size: 80 bytes
0   77 86 fd bc ec 1f 84 71  4d 9f 03 3d ad af 48 3a
1   b2 63 77 44 e3 06 3e b8  65 b7 cc d4 5c db 92 96
2   90 06 43 ce ec c1 12 b3  c9 d1 a7 87 cb de ab 57
3   90 1a a9 13 90 29 78 37  46 9f 26 88 59 b4 b8 1c
4   ff bd 5b b9 92 5b 68 03  3a c3 99 36 6e e8 f8 eb

COMPRESSED DATA SIZE
0   4a 00 00 00
PADDING LENGTH:
0   06
```

Figure 24 – STM1 CLI output for the under-sampling experiment

```
Decrypt and decompress? (y/n): y
PADDING LENGTH CONFIRMATION:
0   06
COMPRESSED DATA SIZE CONFIRMATION:
0   4a 00 00 00
ENCRYPTED DATA CONFIRMATION:
0   77 86 fd bc ec 1f 84 71  4d 9f 03 3d ad af 48 3a
1   b2 63 77 44 e3 06 3e b8  65 b7 cc d4 5c db 92 96
2   90 06 43 ce ec c1 12 b3  c9 d1 a7 87 cb de ab 57
3   90 1a a9 13 90 29 78 37  46 9f 26 88 59 b4 b8 1c
4   ff bd 5b b9 92 5b 68 03  3a c3 99 36 6e e8 f8 eb

DECRYPTED DATA:
Decryption took: 85.69 milliseconds
Decrypted data size: 74 bytes
0   ff 80 0a 01 22 80 41 21  10 88 04 04 02 20 c2 c9
1   40 20 f0 08 58 55 28 d0  08 4c 42 1c 15 69 2c 00
2   12 31 60 aa 5a a0 10 f8  c4 56 01 02 0a 9f fb 70
3   13 18 04 84 2a 98 b8 04  52 41 1d 0a 94 08 2a 54
4   a8 04 7a 01 2b 33 f8 03  c0 16
DECOMPRESSED DATA:
Decompression took: 153.784 milliseconds
Decompressed data size: 132 bytes
0   ff ff ff ff 00 22 00 04  08 08 00 01 00 01 ff ff
1   ff 92 00 07 00 0b ff ff  ff ff 00 46 00 09 10 0e
2   00 01 00 01 ff ff ff 25  00 0e 00 16 ff ff ff ff
3   00 6a 00 0f 18 15 00 02  00 01 ff ff fe b8 00 16
4   00 21 ff ff ff ff 00 8b  00 14 20 1d 00 02 00 02
5   ff ff fe 4a 00 1e 00 2b  00 00 00 00 00 00 00 00
6   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
7   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8   00 00 00 00
CRC32 CHECKSUM:
Average checksum generation time: 0.067 milliseconds
0   d4 7e 8f e6
```

Figure 25 – STM2 CLI output for the under-sampling experiment

STM1 was told to sample 132 bytes of sensor data,. However, it was forced only to sample 88 bytes of data to simulate under-sampling.

The table below shows a comparison between the results from the under-sampling experiment and the results from section 3.1. for compression.

Table XXXIX – Under-sampling experiment compression results

	Uncompressed data size [bytes]	**Compressed data size [bytes]**	**Compression time [ms]**	**Decompression time [ms]**
Original	132	103	189.000	155.795
Under-sampled	132	74	139.111	153.784

As seen from the table above, the under-sampling caused the compressed data to reduce the compressed data size due to more values in the buffer being null. The under-sampling also decreased the compression time because more values were null. However, the decompression remained essentially constant because the algorithm still had to expand the data to 132 bytes.

The table below shows a comparison between the results from the under-sampling experiment and the results from section 3.1. for encryption.

Table XL – Under-sampling experiment encryption results

	Decrypted data size [bytes]	**Encrypted data size [bytes]**	**Encryption time [ms]**	**Decryption time [ms]**
Original	103	104	118.489	117.606
Under sampled	74	80	93.338	85.690

Since the size of both the decrypted data and encrypted data changed, there should be a change in the encryption and decryption time. This behaviour was observed as shown in the table above.

3.5.2. High Sampling Rate Experiment

The CLI output for sampling sensor data at a rate of one sample per 100 milliseconds is shown below.

```
SENSOR DATA:
Sensor data size: 132 bytes
0   ff ff ff ff 00 25 00 04  08 08 00 02 00 01 ff ff
1   ff 93 00 08 00 0c ff ff  ff ff 00 49 00 08 10 0e
2   00 02 00 01 ff ff ff 27  00 10 00 17 ff ff ff ff
3   00 6c 00 0c 18 14 00 03  00 01 00 00 fe bc 00 17
4   00 23 ff ff ff ff 00 8f  00 12 20 18 00 04 00 01
5   00 00 fe 4f 00 1f 00 2e  ff ff ff ff 00 b3 00 19
6   28 1e 00 04 00 01 00 00  fd e3 00 26 00 39 ff ff
7   ff ff 00 d5 00 1f 30 29  00 04 00 01 00 00 fd 76
8   00 2e 00 44
CRC32 CHECKSUM:
0   af 6e 0d 2e
COMPRESSED DATA:
Compression took: 185.24 milliseconds
Compressed data size: 98 bytes
0   ff 80 0a 01 25 80 41 21  10 88 04 0a 01 01 06 16
1   4e 00 09 18 60 55 29 20  91 88 43 82 ad 27 80 44
2   20 11 70 aa 5b 03 a3 18  8a 40 20 61 51 1d 8f fb
3   78 13 18 04 8c 2a 98 f8  04 4a 41 18 22 88 55 29
4   f0 08 fc 02 5c 15 4d 9c  02 33 28 8f 05 57 fb e3
5   80 49 a0 13 90 aa 75 43  e3 30 94 85 5a ec 29 18
6   05 10
ENCRYPTED DATA:
Encryption took: 118.252 milliseconds
Encrypted data size: 104 bytes
0   66 77 e2 5e 55 a3 5e f2  30 33 51 ca c6 8b cb 61
1   ec b6 1e 0c 3f 6b 7c 82  4f bb 49 50 82 b1 98 9d
2   ce 4e b9 b8 db 01 57 fc  16 7b d2 07 1e 60 31 9b
3   a3 a6 e2 e2 3d 2d 3f b8  1d e9 0a 1b e3 bd d9 ea
4   6c cd eb f2 1d 63 db db  f6 16 dd 80 8b ed 08 47
5   cc d2 49 e4 ce 79 c8 db  fb 9d 3c d9 58 7e 03 6e
6   44 57 2f 82 61 cb 31 9a
COMPRESSED DATA SIZE
0   62 00 00 00
PADDING LENGTH:
0   06
```

Figure 27 – STM1 CLI output for increased sensor sampling rate

```
Decrypt and decompress? (y/n): y
PADDING LENGTH CONFIRMATION:
0   06
COMPRESSED DATA SIZE CONFIRMATION:
0   62 00 00 00
ENCRYPTED DATA CONFIRMATION:
0   66 77 e2 5e 55 a3 5e f2  30 33 51 ca c6 8b cb 61
1   ec b6 1e 0c 3f 6b 7c 82  4f bb 49 50 82 b1 98 9d
2   ce 4e b9 b8 db 01 57 fc  16 7b d2 07 1e 60 31 9b
3   a3 a6 e2 e2 3d 2d 3f b8  1d e9 0a 1b e3 bd d9 ea
4   6c cd eb f2 1d 63 db db  f6 16 dd 80 8b ed 08 47
5   cc d2 49 e4 ce 79 c8 db  fb 9d 3c d9 58 7e 03 6e
6   44 57 2f 82 61 cb 31 9a
DECRYPTED DATA:
Decryption took: 112.211 milliseconds
Decrypted data size: 98 bytes
0   ff 80 0a 01 25 80 41 21  10 88 04 0a 01 01 06 16
1   4e 00 09 18 60 55 29 20  91 88 43 82 ad 27 80 44
2   20 11 70 aa 5b 03 a3 18  8a 40 20 61 51 1d 8f fb
3   78 13 18 04 8c 2a 98 f8  04 4a 41 18 22 88 55 29
4   f0 08 fc 02 5c 15 4d 9c  02 33 28 8f 05 57 fb e3
5   80 49 a0 13 90 aa 75 43  e3 30 94 85 5a ec 29 18
6   05 10
DECOMPRESSED DATA:
Decompression took: 157.531 milliseconds
Decompressed data size: 132 bytes
0   ff ff ff ff 00 25 00 04  08 08 00 02 00 01 ff ff
1   ff 93 00 08 00 0c ff ff  ff ff 00 49 00 08 10 0e
2   00 02 00 01 ff ff ff 27  00 10 00 17 ff ff ff ff
3   00 6c 00 0c 18 14 00 03  00 01 00 00 fe bc 00 17
4   00 23 ff ff ff ff 00 8f  00 12 20 18 00 04 00 01
5   00 00 fe 4f 00 1f 00 2e  ff ff ff ff 00 b3 00 19
6   28 1e 00 04 00 01 00 00  fd e3 00 26 00 39 ff ff
7   ff ff 00 d5 00 1f 30 29  00 04 00 01 00 00 fd 76
8   00 2e 00 44
CRC32 CHECKSUM:
Average checksum generation time: 0.067 milliseconds
0   af 6e 0d 2e
```

Figure 26 – STM2 CLI output for increased sensor sampling rate

The table below compares the results from the increased sampling rate experiment and the results from section 3.1. for compression.

Table XLI – Increased sampling rate experiment compression results

	Uncompressed data size [bytes]	**Compressed data size [bytes]**	**Compression time [ms]**	**Decompression time [ms]**
Original	132	103	189.000	155.795
Fast sampling	132	98	185.240	157.531

As seen from the table above, sampling the sensor data at an increased rate decreased the compressed data size from 103 to 98 bytes. This means that more repetitive sequences in the sensor data were found. The compression and decompression took essentially the same length of time to run to completion.

The table below compares the results from the increased sampling rate experiment and the results from section 3.1. for encryption.

Table XLII – Increased sampling rate experiment encryption results

	Decrypted data size [bytes]	Encrypted data size [bytes]	Encryption time [ms]	Decryption time [ms]
Original	103	104	118.489	117.606
Under sampled	98	104	118.252	112.211

As seen from the table above, the increased sampling rate did not significantly affect the encryption algorithm.

3.5.3. Gaussian Noise Experiment

The CLI output for adding Gaussian noise to the sensor data is shown below.

```
SENSOR DATA:
Sensor data size: 132 bytes
0   00 00 ff 00 00 22 00 03  08 08 01 02 01 01 ff ff
1   ff 99 01 08 01 0e ff ff  ff 00 01 46 00 0a 11 10
2   00 01 01 00 ff fe ff 31  01 0f 00 18 ff 00 00 00
3   01 6c 00 10 18 16 01 02  01 00 00 fe fe c8 01 14
4   00 25 ff 00 ff 00 01 8e  01 19 21 1b 01 02 00 01
5   ff fe ff 5f 01 1c 01 32  ff 00 00 ff 01 b0 00 1e
6   29 20 01 02 01 00 ff fe  fd f4 00 22 00 3f 00 ff
7   00 ff 01 d3 00 23 30 26  01 02 01 01 00 fe fd 89
8   00 28 01 4b
CRC32 CHECKSUM:
0   df 33 ed 7f
COMPRESSED DATA:
Compression took: 198.368 milliseconds
Compressed data size: 120 bytes
0   00 0f fc 04 32 28 04 0e  11 08 80 c0 a0 30 1f f8
1   00 73 30 10 48 c3 80 e5  00 80 d1 a0 10 a8 8c 40
2   0c 30 11 08 ff bf f3 18  0c 3e 01 18 14 90 38 6d
3   90 08 84 62 2c 2b 20 50  ff bf dc 88 0c 52 01 25
4   0a 88 ac b1 d0 18 cc 86  36 15 10 40 8a ca bf 01
5   8e 40 66 42 b2 ff c0 76  10 08 f4 a6 40 2b 30 a8
6   ff 7e 86 52 9f 8a 88 2a  3d 38 04 8e 61 26 36 99
7   04 7f b8 98 04 a2 03 4b
ENCRYPTED DATA:
Encryption took: 136.115 milliseconds
Encrypted data size: 120 bytes
0   74 f4 eb 60 60 40 33 60  c9 7d 34 9f cb 0f 02 5c
1   2c 87 d9 43 a6 01 90 ca  af 5d fc 21 ca 53 be 99
2   c3 d0 2a 83 ba f4 0c 29  7d ff 34 82 73 84 d3 d3
3   19 6b 4b 84 9b b2 6b 6d  01 cb 3c 89 44 4d 4a f3
4   a1 32 90 32 52 ad a9 f9  34 7a d1 53 0d 3d 92 f0
5   f6 37 1a 9b 1b 53 68 ce  b3 d1 bb 69 93 b5 2f 36
6   05 e6 5b 1a 33 d6 7f 0d  cc 80 3f fb 0c 21 a1 ef
7   46 7a ff 42 d5 f9 d4 1d
COMPRESSED DATA SIZE
0   78 00 00 00
PADDING LENGTH:
0   00
```

Figure 28 – STM1 CLI output for gaussian noise experiment

```
Decrypt and decompress? (y/n): y
PADDING LENGTH CONFIRMATION:
0   00
COMPRESSED DATA SIZE CONFIRMATION:
0   78 00 00 00
ENCRYPTED DATA CONFIRMATION:
0   74 f4 eb 60 60 40 33 60  c9 7d 34 9f cb 0f 02 5c
1   2c 87 d9 43 a6 01 90 ca  af 5d fc 21 ca 53 be 99
2   c3 d0 2a 83 ba f4 0c 29  7d ff 34 82 73 84 d3 d3
3   19 6b 4b 84 9b b2 6b 6d  01 cb 3c 89 44 4d 4a f3
4   a1 32 90 32 52 ad a9 f9  34 7a d1 53 0d 3d 92 f0
5   f6 37 1a 9b 1b 53 68 ce  b3 d1 bb 69 93 b5 2f 36
6   05 e6 5b 1a 33 d6 7f 0d  cc 80 3f fb 0c 21 a1 ef
7   46 7a ff 42 d5 f9 d4 1d
DECRYPTED DATA:
Decryption took: 134.875 milliseconds
Decrypted data size: 120 bytes
0   00 0f fc 04 32 28 04 0e  11 08 80 c0 a0 30 1f f8
1   00 73 30 10 48 c3 80 e5  00 80 d1 a0 10 a8 8c 40
2   0c 30 11 08 ff bf f3 18  0c 3e 01 18 14 90 38 6d
3   90 08 84 62 2c 2b 20 50  ff bf dc 88 0c 52 01 25
4   0a 88 ac b1 d0 18 cc 86  36 15 10 40 8a ca bf 01
5   8e 40 66 42 b2 ff c0 76  10 08 f4 a6 40 2b 30 a8
6   ff 7e 86 52 9f 8a 88 2a  3d 38 04 8e 61 26 36 99
7   04 7f b8 98 04 a2 03 4b
DECOMPRESSED DATA:
Decompression took: 159.722 milliseconds
Decompressed data size: 132 bytes
0   00 00 ff 00 00 22 00 03  08 08 01 02 01 01 ff ff
1   ff 99 01 08 01 0e ff ff  ff 00 01 46 00 0a 11 10
2   00 01 01 00 ff fe ff 31  01 0f 00 18 ff 00 00 00
3   01 6c 00 10 18 16 01 02  01 00 00 fe fe c8 01 14
4   00 25 ff 00 ff 00 01 8e  01 19 21 1b 01 02 00 01
5   ff fe ff 5f 01 1c 01 32  ff 00 00 ff 01 b0 00 1e
6   29 20 01 02 01 00 ff fe  fd f4 00 22 00 3f 00 ff
7   00 ff 01 d3 00 23 30 26  01 02 01 01 00 fe fd 89
8   00 28 01 4b
CRC32 CHECKSUM:
Average checksum generation time: 0.067 milliseconds
0   df 33 ed 7f
```

Figure 29 – STM2 CLI output for gaussian noise experiment

The table below compares the results from the gaussian noise experiment and the results from section 3.1. for compression.

Table XLIII – Gaussian noise experiment compression results

	Uncompressed data size [bytes]	Compressed data size [bytes]	Compression time [ms]	Decompression time [ms]
Original	132	103	189.000	155.795
Gaussian Noise	132	120	198.368	159.722

As seen from the table above, adding Gaussian noise to the sensor data increased the compressed data size from 103 to 120 bytes and slightly increased the compression time and decompression time. This is because compression looks for repetitive values while compressing to reduce the overall size of the data. Since noise is added to the input data set, fewer values will be repetitive. Hence the algorithm will not be as effective as one with less or no noise in the sensor data.

The table below compares the results from the gaussian noise experiment and the results from section 3.1. for encryption.

Table XLIV – Gaussian noise experiment encryption results

	Decrypted data size [bytes]	Encrypted data size [bytes]	Encryption time [ms]	Decryption time [ms]
Original	103	104	118.489	117.606
Gaussian Noise	120	120	136.115	134.875

As the table above shows, adding Gaussian noise to the sensor data increased the encryption and decryption computation times. The Gaussian noise affected the substitution in the S-boxes; the algorithm took longer to process the data blocks since the data was more random. The same effect occurred with decryption, as the algorithm took longer to process the substituted blocks of data. The compressed size of the data set also increased; this, in turn, also affects the algorithm's speed.

4. Practical Data Acceptance Test Procedures

4.1. Compression ATPs

Table XLV – Practical data ATPs for the compression block

ATP	Description	ATP achieved	Comment
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data. (2 kilobytes per second)	NO	The decompression and compression fails to meet the ATP requirement, this is shown by one of the calculations done to verify the working of the compression algorithm.
Data loss	No data must be lost during the compression of the data file.	YES	The checksum indicates that the entire system is lossless and therefore the compression block must be lossless.
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5.	YES	As indicated in table VII, the compression was at least 1.5 or higher when the rounds were greater than 3.

Compression Time Calculation:

To decompress 1012 bytes, it will take 1122.388ms. This indicates that it will take 2.218s to decompress 2 kilobytes of data – this fails to meet the ATP.

4.2. Encryption ATPs

Table XLVI – Practical data ATPs for the encryption block

ATP	Description	ATP achieved	Comment
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	NO	The encryption and decryption took longer than what was outlined by the ATP requirement.
Data loss	No data must be lost during the encryption and decryption of the data file.	YES	The checksum indicates that the entire system is lossless and therefore the encryption block must be lossless.
Encryption security	The encryption must be strong enough to prevent trivial decryption.	YES	Since the key is static and known only by the client and the STMs. It will only be possible for the client to be able to decrypt.
Encryption integrity	The original data in the file must be identical to the decrypted data in the output file.	YES	The checksum was used to verify the data received by the client following decryption is identical to the data collected by the STM.

4.3. Checksum ATPs

Table XLVII - Practical data ATPs for the checksum block

ATP	Description	ATP achieved	Comment
STM checksum	The STM must create a checksum of the original sensor data successfully.	YES	As can be seen in section 3.4., the checksum algorithm successfully generates a checksum on the first STM.
Client checksum	The client's STM must successfully create a checksum of the decrypted and decompressed data.	YES	As can be seen in section 3.4., the checksum algorithm successfully generates a checksum on the client's STM.
Checksum comparison	The two STMs must generate identical checksums.	YES	As can be seen by the results shown in section 3.1. and 3.4., the checksums are identical

4.4 New Specifications

4.4.1. Compression ATP

Table XLVIII – new specification for compression algorithm vs old specification

ATP	Old Version	New Version
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data. (2 kilobytes per second)	The compression algorithm must not take longer than 1 second to execute on half a kilobyte of data.

4.4.2. Encryption ATP

Table XLIX – new specification for encryption algorithm vs old specification

ATP	Old Version	New Version
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	The encryption algorithm must not take longer than 1 second to execute on half a kilobyte of data.

G. Consolidation of ATPs & Future Plans

1.1. All the ATPs for the entire system

Table L – ATPs for entire system

ATP	Description	ATP achieved
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data. (2 kilobytes per second)	NO
Data loss	No data must be lost during the compression of the data file.	YES
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5.	YES
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	NO
Data loss	No data must be lost during the encryption and decryption of the data file.	YES
Encryption security	The encryption must be strong enough to prevent trivial decryption.	YES
Encryption integrity	The original data in the file must be identical to the decrypted data in the output file.	YES
STM checksum	The STM must create a checksum of the original sensor data successfully.	YES
Client checksum	The client's STM must successfully create a checksum of the decrypted and decompressed data.	YES
Checksum comparison	The two STMs must generate identical checksums.	YES

For the compression and encryption time, it was calculated that the compression and encryption blocks took slightly longer to execute and hence the ATP was not achieved. It was therefore decided that the specifications had to change in order for all ATPs to be met and deem the system a success.

1.2 New Specifications

1.1.1. Compression ATP

Table LI – new specification for compression algorithm vs old specification

ATP	Old Version	New Version
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data. (2 kilobytes per second)	The compression algorithm must not take longer than 1 second to execute on half a kilobyte of data.

1.1.2. Encryption ATP

Table LII – new specification for encryption algorithm vs old specification

ATP	Old Version	New Version
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	The encryption algorithm must not take longer than 1 second to execute on half a kilobyte of data.

1.3 Future Plans

The final product is deemed to be successful and we are happy with what we have produced despite the issues faced. The system works as desired at the time of writing the report and requires no additional work however if it is desired to not need an additional STM microcontroller for decompression or decryption which is external to the SHARC buoy, then a computer with a 32-bit ARM processor is required. There could be modifications for the system to be compatible with a 64-bit ARM processor however it was deemed unnecessary by the team since it was not a requirement for the compression and encryption to take place on the SHARC buoy while the decryption and decompression must exclusively take place on the clients computer; this would require additional features which could be costly and hard to implement in reality. The members of the design team also agreed that the size of storage onboard the STM could be increased in future to allow for larger data sets to be transferred at a time and allow for a stronger encryption. The system as it stands is automated by the use of python software and requires human input before the next step to proceed, this does not require more work however in the future a Graphical User Interface (GUI) can be implemented in order to make the system more user friendly.

The group is very satisfied with implementing both the compression and encryption on the STM microcontroller in the language of C as this is the most power and size efficient language as it does not require additional libraries or plug ins for the system to operate. In the future however, it would be beneficial to test and implement a battery system to determine how long the system will run for and possibly evaluate other methods of decreasing power consumption – as this was not tested.

Since the purpose of this system is to be able to decrease the size of the data and encrypt it for transmission over Iridium (a satellite communication network), the automation of the transmission was not implemented during this phase and would need to be considered before implementation on the SHARC buoy.

H. Conclusion

The team was able to implement compression and encryption successfully on the ARM based STM microcontroller. During the initial phase of analysing the requirements outlined in the brief, it was deemed pertinent by the group that the system be able to compress, encrypt, decrypt, decompress while maintaining efficient power consumption on the STM32F051 Discovery Board. It was also required that the data be collected by the use of a motion tracking device and that at least 25% of the lower Fourier coefficients are maintained; this was achieved by using lossless lightweight encryption and compression algorithms.

Initially it was decided by the group after research into various encryption and compression algorithms that Twofish would be used for the encryption block and LZMA for the compression block. These encryption algorithms were chosen as they were deemed the most reliable and would be most likely to result in a successful system which meets all user requirements. The group had to identify possible shortfalls when making the decision on which algorithm would be used – factors like the fixed-point ARM processor restricting some of the algorithms and the limited memory and RAM. These bottle necks were identified early on to help direct the project towards success.

The requirements made by the client were used to determine functional requirements and design specifications, which would be used to determine the systems outputs as well as functions as a means of providing the client a satisfactory product. The requirements were used to determine the acceptance test procedures which are a means of determining a successful implementation of a sub system and sub-subsystem; an example being the execution time of the algorithm should not exceed 10 seconds per 10 kilobytes of data.

The group did initial analysis and testing with simulated data provided by our tutor, Humphrey Chiramba. The group tested three different data sets, each of different sizes, and different file formats in order to determine if the algorithms chosen could work on the computer – the group opted to use python's built in library for compression and encryption. This was done to gauge the efficiency and speeds of the algorithms and gather data to ensure that the user requirements, acceptance test procedures, and figures of merit have been met before implementation on the STM microcontroller. The results of the simulated data indicated that the chosen algorithms work as desired.

During physical implementation on the STM microcontroller, the group began by interfacing with the SparkFun IMU-20948; this IMU closely resembled what would be used in Antarctica and hence provided a suitable data for testing. It was during this phase of testing that the group identified significant issues with the compression and encryption algorithms. It was concluded that since the algorithms were being implemented in C rather than python and that the processor of the STM being different to the computer used; alternative algorithms must be used. It was then decided to replace Twofish with a modified blowfish algorithm for encryption and LZMA to LZSS for compression. The results following implementation on the STM indicated that the project overall was successful however it also resulted in two of the acceptance test procedures not being met which resulted in revision and consolidation of the ATPs.

Overall the entire project was deemed a great success, despite the limitations faced the group was able to provide a system which met all user requirements and is something the team is proud of making.

I. References

[1] IBM, “Transaction Processing Facility Enterprise Edition,” 05 March 2021. [Online]. Available: https://www.ibm.com/docs/en/ztpf/1.1.0.14?topic=concepts-symmetric-cryptography. [Accessed 17 August 2022].

[2] National Institute of Standards and Technology (NIST), “NIST Technical Series Publications,” 26 November 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf. [Accessed 18 August 2022].

[3] K. Meghna, “Geeks for Geeks,” Geeks for Geeks, 24 February 2022. [Online]. Available: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/?ref=lbp. [Accessed 18 August 2022].

[4] B. Schneier, “Schneier on Security,” [Online]. Available: https://www.schneier.com/academic/blowfish/. [Accessed 18 August 2022].

[5] B. Schneier, “Schneier on Security,” December 1998. [Online]. Available: https://www.schneier.com/academic/archives/1998/12/the_twofish_encrypti.html. [Accessed 18 August 2022].

[6] Geeks For Geeks, “Difference between Lossy Compression and Lossless Compression,” Geeks For Geeks, 8 June 2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-lossy-compression-and-lossless-compression/. [Accessed 17 August 2022].

[7] L. F. Buenavida, “Crunch Time: 10 Best Compression Algorithms,” DZone, 28 May 2020. [Online]. Available: https://dzone.com/articles/crunch-time-10-best-compression-algorithms. [Accessed 17 August 2022].

[8] D. Budhrani, “How data compression works: exploring LZ77,” Towards Data Science, 12 September 2019. [Online]. Available: https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097. [Accessed 17 August 2022].

[9] E. Chen, “Understanding zlib,” January 2019. [Online]. Available: https://www.euccas.me/zlib/#deflate_lz77. [Accessed 17 August 2022].

[10] M. Rodeh, V. R. Pratt and S. Evan, “Linear Algorithm for Data Compression via String Matching,” *Association for Computing Machinery,* vol. 28, no. 1, pp. 16-24, 1981.

[11] J. A. Storer and T. G. Szymanski, “Data compression via textual substitution,” *Association for Computing Machinery,* vol. 29, no. 4, pp. 928-951, 1982.

[12] L. P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3,” May 1996. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1951#section-4. [Accessed 17 August 2022].

[13] Google, “GitHub Brotli,” GItHub, 11 October 2013. [Online]. Available: https://github.com/google/brotli. [Accessed 18 August 2022].

J. Appendixes

I. Raw Results for Table XXX & XXXI

The raw results for all the overall functionality tests are shown in the table below.

Table LIII – Raw results for the overall functionality experiment

Round	Data Size [bytes]	Compressed Data Size [bytes]	Encrypted Data Size [bytes]	Compression Time [ms]	Decompression Time [ms]	Encryption Time [ms]	Decryption Time [ms]
1.1	66	51	56	99.247	81.695	67.316	76.983
1.2	66	53	56	103.082	82.050	67.357	81.268
1.3	66	53	56	100.752	85.665	69.482	77.736
2.1	132	100	104	181.284	159.545	120.919	149.486
2.2	132	97	104	170.482	156.233	120.419	112.123
2.3	132	96	96	164.210	154.892	114.727	111.723
3.1	198	139	144	230.942	229.303	165.888	160.940
3.2	198	143	144	241.268	233.041	163.468	165.672
3.3	198	139	144	233.201	229.045	162.382	158.876
4.1	264	189	192	312.225	305.079	215.970	215.997
4.2	264	176	176	294.273	304.629	197.294	201.156
4.3	264	168	168	277.431	302.546	191.863	194.503
5.1	330	200	200	325.992	377.054	226.650	232.821
5.2	330	207	208	331.882	375.696	231.870	238.365
5.3	330	212	216	349.422	377.768	242.458	241.614
6.1	396	226	232	365.462	448.838	258.948	259.032
6.2	396	227	232	357.379	450.288	257.843	260.137
6.3	396	219	224	352.770	448.132	253.115	250.894
7.1	462	257	264	398.513	525.047	296.662	288.751
7.2	462	252	256	397.882	522.209	287.696	281.429
7.3	462	262	264	417.537	522.628	291.973	289.880
8.1	528	277	280	436.984	596.212	310.615	307.168
8.2	528	269	272	424.623	592.654	301.723	298.265
8.3	528	266	272	414.983	592.631	302.412	295.810

II. Raw Results for Table XXXII & XXXIII

The results for the compression block experiments are shown below in table form.

Table LIV – Raw results for the compression block experiment

Round	Uncompressed data size [bytes]	Compressed data size [bytes]	Compression time [ms]	Decompression time [ms]
1.1	66	52	100.182	80.765
1.2	66	53	100.772	82.757
1.3	66	55	101.176	81.869
2.1	176	135	231.522	208.054
2.2	176	143	236.873	206.914
2.3	176	134	228.199	210.755
3.1	286	185	304.115	327.614
3.2	286	184	308.515	329.998
3.3	286	168	279.246	325.378
4.1	396	229	365.788	447.617
4.2	396	220	348.977	448.724
4.3	396	234	373.962	451.439
5.1	506	259	407.981	568.688
5.2	506	267	425.551	572.181
5.3	506	265	415.255	571.438
6.1	616	322	483.414	690.419
6.2	616	304	468.701	692.684
6.3	616	307	472.839	691.980
7.1	726	357	542.230	814.721
7.2	726	336	516.747	810.182
7.3	726	361	548.701	813.227
8.1	836	416	636.566	932.822
8.2	836	402	618.971	933.682
8.3	836	420	629.237	933.967
9.1	946	432	651.877	1052.466
9.2	946	470	705.835	1055.149
9.3	946	455	679.237	1054.552

10.1	1012	450	675.776	1122.342
10.2	1012	454	682.564	1123.553
10.3	1012	503	763.890	1128.833

III. Raw Results for Table XXXIV & XXXV

The results for the encryption block experiments are shown below in table form.

Table LV – Raw results for the encryption block experiment

Round	Decrypted data size [bytes]	Encrypted data size [bytes]	Encryption time [ms]	Decryption time [ms]
1.1	66	72	85.154	80.893
1.2	66	72	85.679	78.680
1.3	66	72	85.784	81.104
2.1	132	136	153.094	153.286
2.2	132	136	153.089	150.934
2.3	132	136	154.823	149.583
3.1	198	200	226.808	219.817
3.2	198	200	223.974	222.079
3.3	198	200	223.393	220.796
4.1	264	264	295.683	293.328
4.2	264	264	292.465	294.386
4.3	264	264	293.394	294.342
5.1	330	336	371.359	363.825
5.2	330	336	370.470	363.981
5.3	330	336	373.231	367.695
6.1	396	400	441.560	437.072
6.2	396	400	442.391	435.259
6.3	396	400	441.162	435.616
7.1	462	464	509.692	509.616
7.2	462	464	510.034	508.206
7.3	462	464	511.395	507.878
8.1	528	528	580.993	578.576
8.2	528	528	579.247	578.238

8.3	528	528	582.506	580.411
9.1	594	600	661.277	651.804
9.2	594	600	658.119	650.903
9.3	594	600	660.684	653.643
10.1	660	664	728.164	720.719
10.2	660	664	728.972	722.978
10.3	660	664	727.241	721.049