A

Dissertation

On

Evaluation of symmetric key algorithms for Body Sensor Networks (BSN) on Raspberry pi

Submitted in Partial fulfilment of the requirement

for the award of

MASTER OF TECHNOLOGY

(SOFTWARE ENGINEERING)

Submitted By:

Akash Chauhan

Roll No.- 2K12/SWE/02

Under the Guidance of:

Mrs. DivyaShikha Sethia

Delhi Technological University

Department Of Computer Engineering

Delhi Technological University

2012-2014

DECLARATION

I hereby declare that the thesis entitled “**Evaluation of symmetric key algorithms for Body Sensor Networks (BSN) on Raspberry pi”** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree of **Master of Technology in Software Engineering** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

AKASH CHAUHAN
Master of Technology
(Software Engineering)
College Roll No. 2K12/SWE/02
Department of Computer Engineering
Delhi Technological University,
Delhi.

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI-110042

Date: ____________________

This is to certify that the thesis entitled **Evaluation of symmetric key algorithms for Body Sensor Networks (BSN) on Raspberry pi** submitted by **Akash Chauhan (Roll Number: 2K12/SWE/02),** in partial fulfillment of the requirements for the award of degree of Master of Technology in Software Engineering, is an authentic work carried out by her under my guidance. The content embodied in this thesis has not been submitted by her earlier to any institution or organization for any degree or diploma to the best of my knowledge and belief.

Project Guide

Mrs. DivyaShikha Sethia
Assistant Professor
Department of Computer Engineering
Delhi Technological University, Delhi-110042

ACKNOWLEDGEMENT

With due regards, I hereby take this opportunity to acknowledge a lot of people who have supported me with their words and deeds in completion of my research work as part of this course of Master of Technology in Software Engineering.

To start with I would like to thank the almighty for being with me in each and every step of my life. Next, I thank my parents and family for their encouragement and persistent support.

I would like to express my deepest sense of gratitude and indebtedness to my guide and motivator, **Mrs. DivyaShikha Sethia**, Assistant Professor, Department of Computer Engineering, Delhi Technological University for her valuable guidance and support in all the phases from conceptualization to final completion of the project.

I wish to convey my sincere gratitude to **Prof. O.P. Verma,** Head of Department, and all the faculties and PhD. Scholars of Computer Engineering Department, Delhi Technological University who have enlightened me during my project.

I humbly extend my grateful appreciation to my friends whose moral support made this project possible.

Last but not the least; I would like to thank all the people directly and indirectly involved in successfully completion of this project.

Akash Chauhan

Roll No. 2K12/SWE/02

PUBLICATIONS AND COMMUNICATIONS

Paper published in International Conference (IEEE)

Soham Banerjee, Divyashikha Sethia, Tanuj Mittal, Ujjwal Arora, Akash Chauhan, "Secure Sensor Node with Raspberry Pi", IEEE International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT), 2013.

Paper in communication (IEEE Conference)

Akash Chauhan, Kanika Mathur, Divyashikha Sethia, "Evaluation of symmetric key algorithms for Body Sensor Networks (BSN) on Raspberry pi", IEEE INDICON, 2014.

TABLE OF CONTENT

DECLARATION……………………………………………………………………...……ii

CERTIFICATE………………..……………………………………………….…..….iii

ACKNOWLEDGEMENT…….....………………………………………………….....iv

PUBLICATIONS AND COMMUNICATIONS………..……………………………………..v

TABLE OF CONTENTS……………………………………………………………...vi-vii

LIST OF FIGURES…..………………………………………………………………..viii

LIST OF TABLES…………………………………………………………………...ix

LIST OF GRAPHS……………………………………………………………………x

ABSTRACT……………………………………………………………………………..xi

CHAPTER1: INTRODUCTION…………………………………………….........................1

1.1. MOTIVATION OF THE WORK...2

1.2. PROBLEM STATEMENT 2

1.3. ORGANIZATION OF THE THESIS..3

CHAPTER 2: SURVVEY……………………………………………………………..5

2.1 CRYPTOGRAPHY…………………………………………………………….5

2.2 LITERATURE SURVEY……………………………………………………………6

CHAPTER 3: PROTOTYPE……………………………………………………….....9

3.1 RASPBERRY PI USED AS A PROTOTYPE………………………………………..9

CHAPTER 4: IMPLEMENTATION…………………………………………………11

4.1. PREVIOUS WORK...11

4.2. PRAPOSAL 12

4.3. IMPLEMENTING BODY SENSOR NETWORK (BSN) ALGORITHMS ON RASPBERRY PI…………………………………………….....................................13

4.4. IMPLEMENTATION OF COMPARISON FACTROS 18

4.4.1. EXECUTION TIME...18

4.4.2. MEMORY OCCUPIED AT RUN TIME...19

4.4.3. LOC 19

4.4.4. CONTEXT SWITCHES PER SECOND (CSS)..20

4.4.5. PAGE FAULTS PER SECOND (PFS) 20
4.4.6. CPU USAGE 20
4.4.7. CPU TIME 21
4.4.8. CACHE MISS RATE 21
4.4.9. READ OPERATION PER SECOND (ROS) 21
4.4.10. WRITE OPERATION PER SECOND (WOS) 22
4.4.11. POWER CONSUMPTION 22

CHAPTER 5: TESTING AND RESULTS 24
5.1. EXECUTION TIME 24
5.2. MEMORY OCCUPIED AT RUN TIME 26
5.3. LOC 27
5.4. CONTEXT SWITCHES PER SECOND (CSS) 28
5.5. PAGE FAULTS PER SECOND (PFS) 29
5.6. CPU USAGE 30
5.7. CPU TIME 31
5.8. CACHE MISS RATE 32
5.9. READ OPERATION PER SECOND (ROS) 34
5.10. WRITE OPERATION PER SECOND (WOS) 35
5.11. POWER CONSUMPTION 36

CHAPTER 6: ISSUES WITH BSN ALGORITHMS 38
6.1. SPECIAL CASE OF SENSOR NODES 38
6.2. ISSUES WITH THE ENCRYPTION ALGORITHMS 38
6.3. SITUATION BASED USE 40

CHAPTER 7: CONCLUSIONS AND FUTURE WORK 43
7.1 CONCLUSIONS 43
7.2 FUTURE WORK 45

REFERENCES 46
APPENDIX 50

List of Figures

Figure 2.1: Hierarchical tree depicting types of cryptography algorithms……………………………6

Figure 3.2: The Raspberry pi……………………………………………………………………………10

Figure 4.1 Flow diagram of body sensor [15]…………………………………………………..........11

Figure 4.2: A basic prototype of the implementation of BSN using the Raspberry pi………….……12

Figure 4.3: Feistal cipher structure for RC5 [39]……………………………………………………15

Figure 4.4: Blowfish Encryption [36]…………………………………………..………………16

Figure 4.5: G permutation function [37]…………………………………………..…………17

Figure 4.6: Setting rules for Skipjack (Pictorial representation) [37]……………………..............17

List of Tables

Table 3.1: Comparison of various single board computers [15]……………………………………...10

Table 5.1: Execution time for various algorithms (in ms) on variation of accelerometer reading…...24

Table 5.2: Memory occupied by various algorithms (in bytes)………………………………………26

Table 5.3: Lines of Code (LOC) for various algorithms……………………………………………..27

Table 5.4: Context switches for various algorithms on variation of accelerometer readings………...28

Table 5.5: Page faults per second for various algorithms on variation of accelerometer readings…..29

Table 5.6: CPU Usage for various algorithms on variation of accelerometer readings……………...30

Table 5.7: CPU time for various algorithms (in ms) on variation of accelerometer readings………..31

Table 5.8: Cache miss rate for various algorithms (ms) on variation of accelerometer readings.........32

Table 5.9: Read operations per second for various algorithms (in ms) on variation of accelerometer readings…………………………………………………………………………………………….34

Table 5.10: Write operations per second for various algorithms (in ms) on variation of accelerometer readings…………………………………………………………………………………………….35

Table 5.11: Power consumption for various algorithms (in watts) on variation of accelerometer readings…………………………………………………………………………………………….36

Table 7.1: A complete summation table of all the results obtained…………………………………..43

List of graphs

Graph 5.1: Execution times verses Accelerometer input for RC4, RC5 and Skipjack……………….24

Graph 5.2: Execution times verses Accelerometer input for Blowfish……………………………25

Graph 5.3: Memory occupied verses algorithm……………………………………………………26

Graph 5.4: Lines of code verses algorithm………………………………………………………...27

Graph 5.5: Context switches per sec verses Accelerometer input for RC4, RC5, Skipjack and Blowfish……………………………………………………………………………………………28

Graph 5.6: Page faults verses Accelerometer input for RC4, RC5, Skipjack and Blowfish………..29

Graph 5.7: CPU Load verses Accelerometer input for RC4, RC5, Skipjack and Blowfish………….30

Graph 5.8: CPU time verses Accelerometer input for RC4, RC5 and Skipjack……………………...31

Graph 5.9: CPU time verses Accelerometer input for Blowfish……………………………………..32

Graph 5.10: Cache miss rate verses Accelerometer input for RC4, RC5, Skipjack and Blowfish……………………………………………………………………………………………33

Graph 5.11: Read operations per second verses Accelerometer input for RC4, RC5, Skipjack and Blowfish……………………………………………………………………………………………34

Graph 5.12: Write operations per second verses Accelerometer input for RC4, RC5, Skipjack and Blowfish……………………………………………………………………………………………35

Graph 5.13: Power consumption verses Accelerometer input for RC4, RC5, Skipjack and Blowfish……………………………………………………………………………………………36

ABSTRACT

BSNs (Body sensor networks) have been widely implemented in medical environments. Monitoring the patient health and helping in fast and timely data access and transfer, BSNs pose a major reform in the field of medicine. There have been various techniques through which BSNs have been implemented, one being that using the Raspberry pi, a single board computer (SBC), which presented a new milestone in the field of sensor networks [15]. The data being sent over these sensor devices is critical and with increasing software attacks, comes the major challenge of providing a secure data transmission.

The BSN implementation using the Raspberry pi incorporates the RC 4 encryption to establish a secure communication. But situations have been witnessed where RC4 does not suffice to the needs of the situation and in fact has detrimental overheads on the system.

In this work, various symmetric key cryptography algorithms used in the BSNs has been segregated namely RC4, RC5, Blowfish, and Skipjack. An implementation of these algorithms has been carried out on the Raspberry pi. Further using available LINUX packages/utilities, a comparison of these algorithms has been laid forth as to find the most suited algorithm in a given situation.

Also by simulating the values obtained for the algorithms on MATLAB, a proper graphical analysis of the entire work has been depicted. A situational comparison of these algorithms along with their weaknesses shown will help in segregating the use of these algorithms according to the need of the time and situation.

Index Terms—BSN, encryption, Blowfish, RC4, RC5, Skipjack, Single Board Computer, Raspberry pi, Accelerometer, LINUX package/utilities

CHAPTER 1: INTRODUCTION

With the ever increasing incorporation of technology into healthcare, and with advances being made regularly, one can imagine the future time nursing homes or hospitals running on pervasive networks that can not only provide a continuous medical monitoring , but also a medical data access and also emergency communication. The patient condition monitored through sensor devices, can be sent to a doctor, who can prescribe the appropriate medication to be provided. The diagnosis is done primarily through BSN (Body Sensor Network). A BSN is a wireless ad hoc network that has sensors that are attached to a patient body and also medical devices kept in close proximity. The data collected can further be transmitted to a doctor’s PDA who can take necessary actions as and when needed. Such sensor devices, called body sensor devices, not only help improve the doctor-patient efficiency, but the use of wireless technology enables doctors to monitor patients remotely and give them timely health information, reminders, and support as and when required – thereby extending and improving the reach of health care and thus making it available anywhere, anytime and for anyone. Databases are created containing all patient records from which authenticated users can access the records and at times of emergency, the patients can be rendered healthcare facilities even remotely.

Various implementations of BSNs have been proposed. In our earlier work we have a proposed a prototype of a Body Sensor using a Raspberry Pi [15] in which various sensors can be attached to the pi to monitor vital health parameters of a patient.

As the case of sensor devices, data being sent and accessed remotely, any tampering with patient statistics can lead to dangerous outcomes. Thus data security becomes an issue of prime concern. Thus we need an efficient crypto system so as to securely transmit data.

For this we first analyze the basic cryptography algorithms. The use of an algorithm for BSN is critical as these devices have constraints such as low processing power, less memory and limited bandwidth. The algorithms are primarily categorized into two types, symmetric key and asymmetric key. The various cryptography algorithms used in Body Sensor Networks include RC4 [7], RC5 [2] etc.

But each algorithm has its own advantages and disadvantages. In the implementation of a BSN using the raspberry pi, the entire model was made to use RC 4 for data encryption [15]. But in not all situations, do we find RC 4 to be the best suitable algorithm. Some researchers agree on the fact that RC5 is the best algorithm for use, but this is not always true [11].

Research comparison had done only on basis of space and time complexities are not enough to predict the suitability of an algorithm. There are a lot more factors which can help determine the feasibility of the use of the algorithm in Body Sensor Networks.

In our work, various symmetric key encryption algorithms used in BSNs have been segregated. The reason behind using symmetric key algorithms is that since BSNs are low computation devices, they need algorithms involving less amount of computation work which is provided by these algorithms. Further a comparison concerning various factors of performance such as execution time, CPU usage, Cache miss rate, Page faults has been carried out. Later discussing the issues relating to the algorithms and determining situations where these algorithms suit best, we present what is called situational use of the algorithms.

1.1. MOTIVATION OF THE WORK

As we know that security in data communication is a major issue to be considered. A security work has been done [15] to make sure about security will retain in high priority in data communication. Many cryptographic algorithms can be used to transfer data from Raspberry pi to Android Based Phone. So RC4 algorithm [7] has been used to retain security in communication [15]. To retain security in communication very few factors like time and space have been considered in this work [15]. But wouldn't be right to say that RC4 [7] is ideal algorithm for data communication or for all the situations. So this thing motivated us. Hence we have done with analytical survey on various conferences and journals to find out what are the most suitable algorithms for BSN. Then we found some other algorithms should be considered in the picture. Now after all we have done with implementation of such algorithms on Pi and then many comparison factors have been considered to compare these algorithms while sensor readings attached to Raspberry pi are varying from 100 to 2000.

1.2. PROBLEM STATEMENT

From the beginning section we can conclude that security is the major issue to be considered in data communications. As we have discussed above in motivation section also, a security work has been done [15] to make sure about security will retain in high priority in data communication. Hence RC4 algorithm [7] has used for this work [15]. But we can't say that RC4 is the only algorithm in BSN which is most suitable for all the situations. Many cryptographic algorithms are generally being used in BSN. Hence after a statistical survey on various journals and conferences, some cryptographic techniques have been found which are most suitable for BSN. To know which algorithm is suitable for which situation a comparison of these algorithms for possible comparison factors is needed to be done when a body sensor device is connected to Raspberry pi and the readings of this sensor device are varying from 100 to 2000. In the proposed work we have implemented all these algorithms on Raspberry pi and compared these algorithms according to some important comparison factors while Accelerometer readings are varying. To understand the whole in a simple way the graph comparison is need to be done. Hence MATLAB simulation work has also been done for graph comparison. With graph simulation it's easy to understand what actually is varying with what.

1.3. ORGANIZATION OF THE THESIS

This thesis is organized as follows:

Chapter 2 discussed about cryptography and security factors. This includes the extensive study on various BSN's (Body Sensor Networks) algorithms that have been proposed in the literature so far. It also highlights some of the most relevant works in the direction of field of work presented in the thesis.

Chapter 3 discussed about Raspberry pi. This chapter also highlights various applications of Raspberry pi, why we have chosen it for our work and why Raspberry pi is suitable SBC (Single board Computer) among all.

Chapter 4 discussed about Implementation. It includes brief about proposed work, Implementation of various BSN algorithms, various Linux based Utilities/Commands have been included and implementation of all the parameters on Raspberry pi.

Chapter 5 presents a detailed analysis of the results obtained. This chapter also depicted the results of comparison. It includes details about each individual result of comparison, tables have been used to compare BSN algorithms and Matlab simulation has also been carried out to generate comparison graphs.

Chapter6 discussed about Special case of sensor nodes. This chapter also highlights various Issues with the encryption algorithms and situation based use of these algorithms also highlighted.

Chapter 7 presents the conclusions, summarization of the thesis and future work.

CHAPTER 2: SURVEY

2.1. CRYPTOGRAPHY

Cryptography is defined as a technique of encrypting data at sender side and converting it into cipher text using some algorithm (Encryption). This data then travels over the channel and at receiver side the cipher text is converted back into plain text using the same algorithm (Decryption). The plain text conversion into cipher text is the most important as this is what travels over the channel. Better the strength of the encryption algorithm, less are the chances of the intruder being able to decode the cipher text.

The security of data is bound to five factors: [5]

A) Authentication: Refers to assuring the receiver that the sender is actually an authenticated one.
B) Data Integrity: Data received is same as that sent by the sender.
C) Access Control: Prevention of unauthorized access of resources.
D) Data Confidentiality: Keeping data secret with no outside access.
E) Non-Repudiation: Provides protection against denial by one of the entities involved in communication.

All these factors should be taken into care while designing a technique for the security of data. The case of BSN becomes all the more crucial since we have additional factors to be taken into concern as they are low power devices and carry sensitive information. We have limited storage, bandwidth and energy [1]. Also cost is a major issue that needs to be addressed.

Thus we have analyzed the algorithms that are used for data security in sensor devices and presented a detailed comparison. Also the algorithms have been separated depending on their area of use in today's time.

2.2. LITERATURE SURVEY

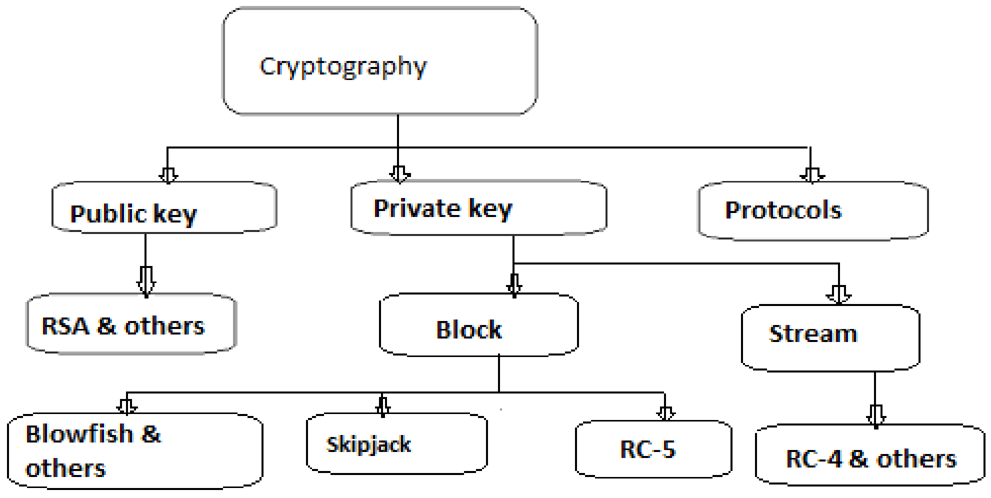


Figure 2.1: Hierarchical tree depicting types of cryptography algorithms

A survey was conducted in the data security field to find out various cryptography algorithms used in sensor devices and specifically those used in BSN. Summarizing the survey, we conclude cryptography to be mainly divided into three types (Figure 2.1):

i) Public Key cryptography
ii) Private Key cryptography
iii) Cryptography using Protocols.

The most popular Public key cryptography techniques are:

i) RSA [6]
ii) ECC [10]
iii) ECDH(Elliptic curve Diffie-Hellman) [10]
iv) IBE-Lite [10]

Private Key cryptography further divided into Block encryption and Stream encryption.

The most popular Block encryption techniques

i) RC4 [11]
ii) RC5 [2]
iii) Blowfish [8]

iv) Triple DES [8]

v) Skipjack [14].

Stream encryption technique includes RC4 [7]. Among cryptography protocols are SSL and TLS. SSL and TLS use RC4 Algorithm [7] in web browsing.

Public key encryption methods have not been much successful on bio sensor nodes as these devices have a low computation power [10]. The new Public key encryption algorithm developed IBE-Lite is also not so suitable for BSN since no proper authentication is provided and it is susceptible to node duplication attacks [10]. Despite its use in BSN's today, Public key encryption methods are not so desirable for the BSN as separate keys are required which makes implementation difficult on these low power and computation devices.

Moving to Private Key encryption (symmetric key), it is widely used for BSN; however symmetric key ciphers can also be expensive to implement on some target platforms. Private key cipher(- divided into Stream and Block Cipher), Stream ciphers (RC 4, RC5 etc.) have a simple architecture and a fast encryption rate and are thus considered more suitable for sensors having limited memory and computing resources as compared to block ciphers (AES, DES etc) [6].

Various research works indicate that the most commonly used algorithm for BSN is RC-5. According to the Gawali and Wadhai 2012, the RC5 Algorithm [2] is good choice on the basis of its overall performance [2]. Amini, Verhoeven, Lukkien and Chen 2011, considered that the RC4 algorithm [7] is a good option to use for small size messages in BSN. RC5 can be considered as one of the best ciphers in terms of overall performance, when used in nodes with limited memory and processing capabilities [2]. But various tradeoffs to this exist. Situations having message/input data not large enough, use RC4 [7], whereas those needing block encryption, use RC5 [2].

Also other algorithms include DES, Triple DES [8], AES [2], Blowfish, Skipjack, SHA 1(secure hash algorithm), MD5 (for message digests), A5, RC-4 [7] and RC- 6. AES [2] is also widespread, having inbuilt hardware support for some platforms. Comparing the energy consumption of AES, RC-5 and DES, we find RC-5 to consume lower energy [2]. RC5 is better than DES in security strength and implementation efficiency [2]. RC-4 and Skipjack can be used to provide lightweight message confidentiality for our security system [7].

Elminaam, Kader, and Hadhoud 2010, conclude that Blowfish [8] has a comparatively better performance than AES [2], DES [2], Triple DES [8], RC6 and RC2.

Additionally Daniel Tze Huei Lai, Rezaul Begg, Marimuthu Palaniswami in their publication- Healthcare sensor networks challenges toward practical implementation stated that Skipjack algorithm is also in great use in BSN [14].

Skipjack is a symmetric key cipher and its use in BSN has increased in the recent times. It is very efficient in terms of security.

CHAPTER 3: PROTOTYPE

3.1 RASPBERRY PI USED AS A PROTOTYPE

Raspberry pi is a commonly used Single Board Computer (SBC) in many applications such as: [3]

1. Web server (-LAMP-Linux Apache Mysql PHP)
2. Household appliance control center(its automation)
3. 3G modem or GPS system
4. Arcade Game
5. Audio Book player
6. Super computer
7. Configure green energy charging devices
8. Transmit messages from a mobile device to a printer or appliance
9. Wearable computer
10. Body sensor devices.

As far as BSN are concerned, raspberry Pi offers a good choice. We also have other Single Board Computers available such as Beagle Bone and Panda Board. Similar to the Raspberry Pi, both are exposed boards with ARM processors with HD video capability. But we prefer using Raspberry Pi in body sensor devices due to several reasons [4]:

A) The Raspberry Pi is a very compact and portable device. It can be assumed to be a complete working computer. Inserting SD card containing the OS, and connecting the peripherals and power, it becomes ready to use. Beagle Boards and Panda Boards require hookup to a host computer for initial setup, and though they have similar processing capabilities, they take a little more know-how to get them fully functional.

B) The area it requires is least among all the Single Board Computers and uses an average RAM

C) Also keeping the cost factor as of prime importance, Raspberry Pi is the best possible choice for developing our Body Sensor Device.

A comparison is shown between the Raspberry pi and other Single Board Computers table-3.1.

Table 3.1: Comparison of various single board computers [15]

Features	BeagleBone	Raspberry Pi	Panda Board
Single-core/Multi-core frequency	Single-core/ 720 MHz	Low power single-core/ 700 MHz	Dual-core/ 1 GHz
Area	45 cm^2	45 cm^2	115 cm^2
RAM	256 MB	512 MB	1 GB
Price	$133	$35	$500

Figure 3.2: the Raspberry pi

As we have discussed above that Raspberry pi is a good choice among all Single Board Computers. We can use Raspberry pi as a prototype for this kind of work. Raspberry pi can be attached to various Body Sensor Devices like Accelerometer for fall detection, Thermometer for temperature measurement, etc. A python Script can be useful to sense data from these Body Sensor Devices [15]. Also it has been implemented on Raspberry pi [15] to stream the sensor readings to the medical professional's mobile device through communication interfaces like Bluetooth, NFC etc.

Also security in Body Sensor Networks (BSN) is of top priority since the patient data should be preserved and should not be accessed by any unauthorized person.

Various encryption algorithms have been implemented for secure data transmission by the Raspberry pi. The main issue encountered with the sensor devices is that the entire system needs to be cost efficient. In the task to make a secure connection between the Raspberry pi and the Android Based Phone, we need a time efficient algorithm. Various researches have been carried out to device such algorithms.

CHAPTER 4: IMPLEMETATION

4.1. PREVIOUS WORK

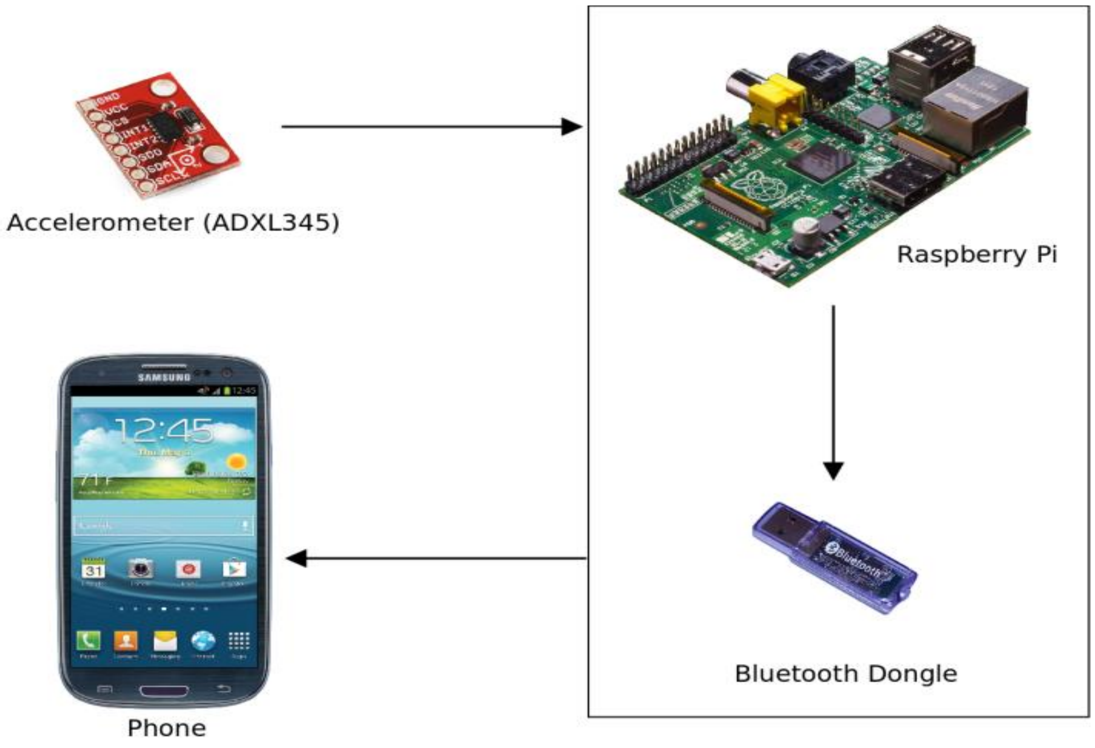


Figure 4.1: Flow diagram of body sensor [15]

The figure 4.1 depicts a basic prototype of the implementation of BSN using the Raspberry pi [15]. Figure 4.1 contains an accelerometer (connected to the Raspberry pi) which is a BSD (Body Sensor Device) that gives the position of a person in the 3D co-ordinate system. The function of accelerometer is to keep track of fall detection for elderly people. A python script is written which contains code for stabilizing the automated Bluetooth connection between the Raspberry PI and android based phone. A Bluetooth module was used for this. The python script is basically helps in the automation task. The python script also accesses data from the accelerometer and correspondingly saves it onto the Raspberry pi board [15]. Succinctly the aim was to get the accelerometer readings on android based phone. Now as the implementation task is completed, care must be taken that the data being send from the Raspberry pi to android based phone must be secure. Also any unauthorized access is undesirable. So an efficient cryptography algorithm was needed. A survey was carried out to find which algorithm is most suitable for this work (keeping time and space as the main constraints). On the basis of the survey, RC4 algorithm was concluded to be the best choice [15]. But even later, owing to a few more constraints taken into consideration for sensor devices such as power or CPU usage, the conclusion of RC 4 being the best came into doubt. Situations were witnessed where few other algorithms fared better than RC 4. Thus a need was felt for algorithms to be segregated depending on the feasibility, given constraints of the system.

Thus the work was initiated with a survey to find out the most popular algorithms being used in BSN. Results obtained included RC4, RC5, skipjack and blowfish (focus was laid only on symmetric key algorithms as they offer comparatively low computation as is required by sensor devices). Later a comparison was laid forth considering these algorithms on the basis of various constraints such as power consumption, CPU usage and Cache miss rate.

4.2. PROPOSAL

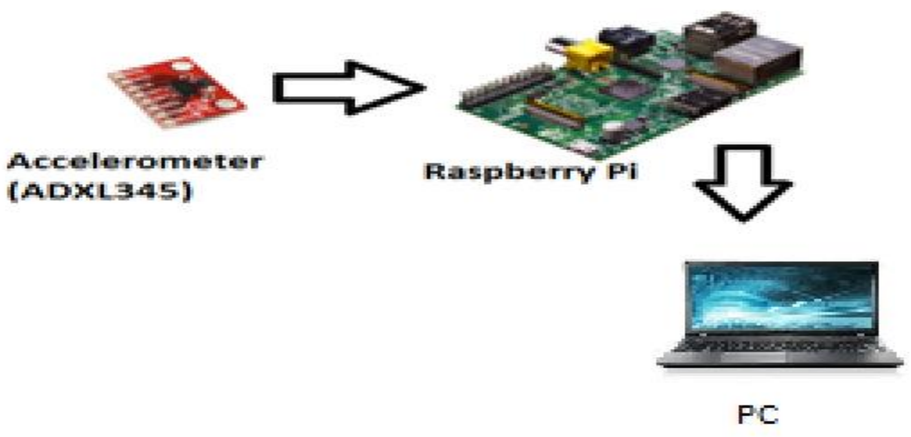


Figure 4.2: A basic prototype of the implementation of BSN using the Raspberry pi

The Accelerometer senses the readings of the Pi, which are accessed by the pi using python-scripts (written on it while programming it) [15]. After the execution of these scripts on pi, the data is stored in certain text files. (Refer Figure 4.2)

Three such text files have been considered, which include 10, 100 and 1000 readings from the accelerometer respectively. Now, beginning with the comparison work:

- An Accelerometer Body sensor device is connected to Raspberry pi and a python script helps us to sense sensor readings and then store these readings into text file [15].
- These readings have to be communicated out through Bluetooth to a mobile device in a secure manner. For this purpose various encryption algorithms like RC4, RC5, Skipjack and Blowfish have implemented in C language.
- First we have implemented all these algorithms on Raspberry pi. Here Raspberry pi is considered as a prototype only. Then all the comparison factors have implemented for all these algorithms and vary size of sensor readings.
- Both encryption and decryption are being deployed on Raspberry pi only because we wanted to check the performance of these algorithms for different comparison factors and sensor readings are varying, when Raspberry pi is being used as a prototype only.
- The performance of individual algorithms is then stored in text files for different size of input data and for different comparison factors.
- Furthermore the various factor comparisons, results and statistics have been simulated on MATLAB to present a better distinction between the algorithms.

Thus the algorithms have been compared and the best has been recommended according to given resources and situation.

Also a situational comparison has been put forth wherein the algorithm best suited in a particular scenario (apart from their use in body area network) has been shown. This will provide ease of use as to which algorithm should be implemented under a given situation.

4.3. IMPLEMENTING BODY SENSOR NETWORK (BSN) ALGORITHMS ON RASPBERRY PI:

After a detailed survey of the research work carried out in the field of BSNs, the cryptography algorithms used have been segregated as RC4, RC5, Skipjack and Blowfish. As discussed earlier a Small Board Computer (SBC), Raspberry pi has been used as a prototype for the work. All these algorithms have been implemented on the Raspberry pi. The major drawback in case of the Raspberry pi is its instability; the operating system may crash if proper precaution is not taken. Raspbian Operating system has been used for this work. It is slightly different from Ubuntu, the similarity being that both are Linux based. All the algorithms have been coded in C Language on Raspberry pi. Our main aim was to measure the performances of these algorithms for varying sizes of Accelerometer input. Once these algorithms have been implemented, their performance has been measured for various comparison factors given in chapter 5.

The various security algorithms implemented are:

A) **RC-4**: RC-4 is a stream cipher which uses byte oriented operations. The data stream undergoes XOR together with a series of generated keys. The output is then XOR-ed together with the stream of data in order to generate a newly-encrypted data.

Pseudo code for RC4 [33]

The Pseudo code for RC4 is included in appendix 5.

Working [34]:

A variable-length key (ranging from 1 to 256 bytes) initializes a 256-byte state vector S, with elements S [0], S [1], …, S [255]. For encryption and decryption, a byte k is generated from S by choosing one of the 255 entries in a systematic manner. With each generated value of k, entries in S are once again permuted. For the purpose of encryption, the value of k is XORed with the next byte of plaintext. For decryption, XOR the value k with the next byte of cipher text.

B) **RC-5:** RC-5 is a symmetric key block cipher. It has variable block sizes such as 32, 64 or 128 bits. Its encryption and decryption routines are like Feistal cipher structure involving XOR and modular addition operations.

Pseudo code for RC5 [38]:

The Pseudo code for RC5 is included in appendix 6.

Working [38]:

The input to RC5 consists of two w-bit words which denoted by A and B.

RC5 uses an expanded key table S[0...t-1], consisting of t = 2(r-1) w-bit words, where r is the number of rounds. The first step is Key Expansion which expands the user's secret key K to fill the expanded key array S. The algorithm uses two magic constants and consists of three simple algorithmic parts –defining the magic constants (Pw and Qw), conversion of the Secret Key from Bytes to Words, initialization of the Array S. Later the secret key is mixed followed by encryption.

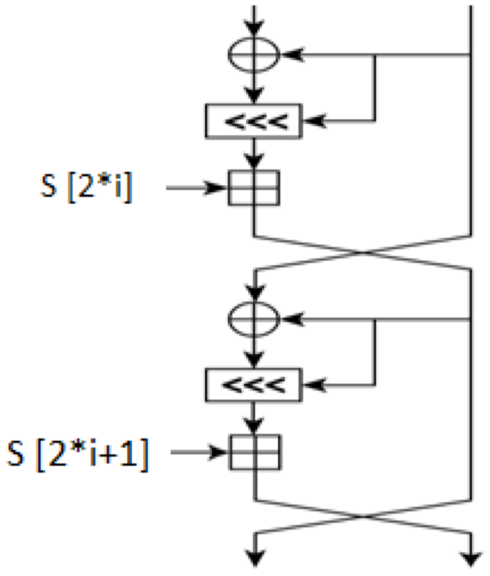


Figure 4.3: Feistal cipher structure for RC5 [39]

C) **Blowfish:** Blowfish is a symmetric key block cipher, having 64 bit block size using variable length key size. It also follows Feistal cipher structure with 16 rounds and key dependent S boxes.

Pseudo code for Blowfish [35]:

The Pseudo code for Blowfish is included in appendix 8.

Working [35]:

The Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time. It follows the feistel network and main working of the algorithm can be divided into two parts namely Key expansion and data encryption.

The key expansion part converts a key of at most 448 bits into several sub key arrays. Blowfish uses large number of sub keys and these keys are generated before any data encryption or decryption. Total, 521 iterations are required to generate all sub keys.

The data encryption has a function to iterate 16 times in a network. Each round has key-dependent permutations and a key and various data-dependent substitutions. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookup tables for each round.

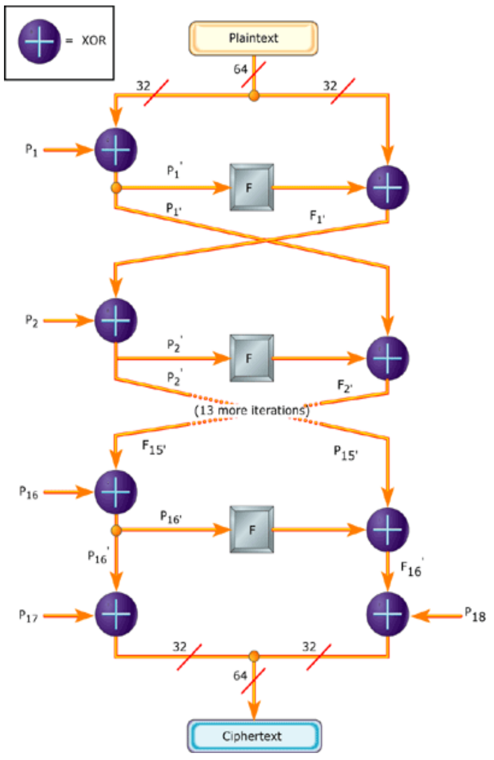


Figure 4.4: Blowfish Encryption [36]

D) **Skipjack:** Skipjack is a block cipher. A symmetric key encryption algorithm, it does encryption or decryption of 64 bit data blocks using an 80-bit key. It uses an unbalanced Feistal cipher structure with 32 rounds.

Pseudo code for Skipjack [37]:

The Pseudo code for Skipjack is included in appendix 7.

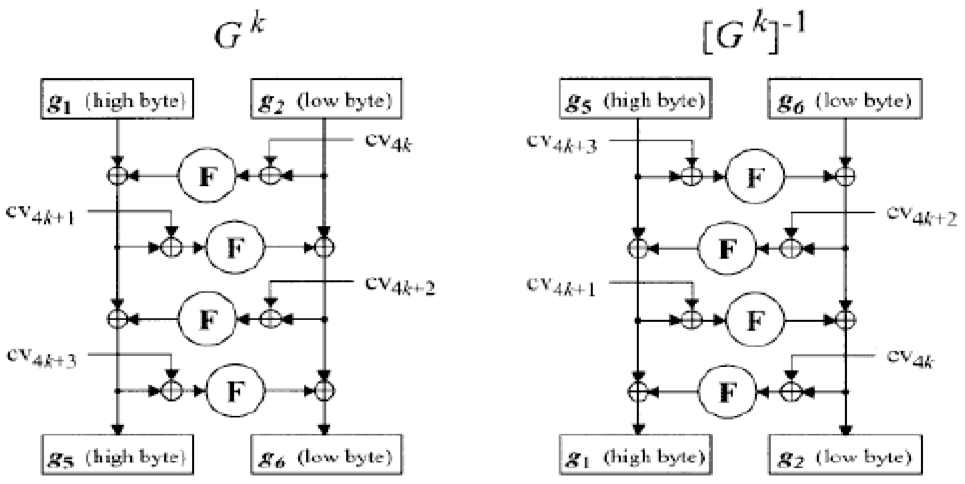


Figure 4.5: G permutation function [37]

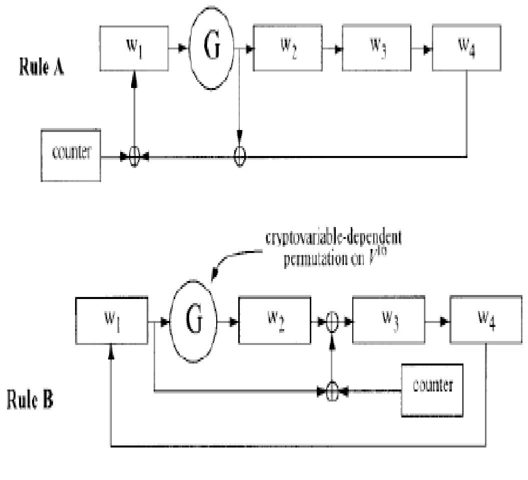


Figure 4.6: Stepping rules for Skipjack (pictorial representation) [37]

Working [37]:

The SkipJack algorithm uses an 80 bit key which is broken up into 10 bytes, with four bytes used in each round. SkipJack iterates through 32 rounds, 8 rounds of round A, 8 of round B and then repeats the same loop again. Each round permutes the 64-bit plain text and passes 16-bits of it through a substitution function (the G function). The G function uses a static 8 bit substitution table (the F table) and a Feistel cipher structure to manipulate the input bits further. All functions used are reversible, and SkipJack decryption is carried out in just the opposite way. To decrypt a cipher text, the key schedule is reversed and the order of the rounds and all the arrows in the Feistel structure are also reversed. The plain text is obtained from the cipher text using just the opposite procedure.

4.4. IMPLEMENTATION OF COMPARISON FACTORS:

The main difficulty encountered in the entire work that it is comparatively easy to measure the performance factors for process while the same task poses challenges in case of processes. This is because program being passive entities run only for a very small fraction of time, so calculating all performance metrics in that short time span is a bit difficult. Several tools and commands can be used to measure performance of a process, but it not same for programs. If we have to check the performance of a program, then various kernel level commands/packages have to be used. Thus we have calculated the performance metrics for a program by working with several kernel level package/tool/command. Now we describe the procedure used to calculate each performance metric, various packages and commands used have also been described.

4.4.1. EXECUTION TIME:

C language has a function named clock () used to measure the current time of the system clock. The same has been used to measure times before and after the program execution. The difference in values of two clock () functions gives execution time for a program. The time.h library and Clock_t data type have been used to incorporate the clock () function into the program.

```
#Clock_t start;
#..........
#...........
#...........
#...........
#Clock_t finish;
#Time= (float) (finish-stop)/CLOCKS_PER_SEC;
```

In the above example, 'Time' indicates the amount of time required by the portion of the program (in seconds). The same has been repeated for all algorithms, varying the accelerometer inputs from 100 to 2000.

This gives the final execution time for various algorithms.

4.4.2. MEMORY OCCUPIED AT RUN TIME:

Linux based utility has been used to measure the amount of memory occupied at run time for a program. The "size" command is used giving the final output in bytes. The executable file name of a program has been used along with the size commands. It looks like this on terminal:

```
#size "executable file name of program"
#size ./abc
```

**The same has been repeated for the executable files of all algorithms. This finally gives the amount of memory occupation. Memory Consumed (MC) at run time does not vary with the Accelerometer Input.

4.4.3. LOC:

Linux based utility have been used to find out number of lines in a program. The "nl" command gives information about how many lines of code a program contains. As discussed earlier, all algorithms have their code in C. While using the nl command, the source file name along with its extension is added. It looks like this on terminal:

```
#nl "source code file name with extension"
#nl abc.c
```

** The above code returns the number of lines of code a program has used. We did this for all the algorithms. Thus we obtain the number of lines of code used by RC4, RC5, Skipjack and Blowfish algorithm. Lines of Code (LOC) don't vary with Accelerometer Input.

4.4.4. CONTEXT SWITCHES PER SECOND (CSS):

Once again Linux based utility commands have been used to find out the number of Context Switches occurring per Second in a program. Colin King, a well known kernel Engineer, first introduced this linux based package named health-check. [28] We have also taken reference from git clone [30] to install this package on the Raspberry pi. Before the installation of this package, another package was installed namely libjson0-dev. Thus the sequence of package installation is libjson0-dev first and then health-check. After completing the installation of packages, the following commands were run on the terminal:

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In the above example abc is the executable file. The above commands generate a lot of information, but our main focus here is on the Context Switches occurring per second for a program. The same has been repeated for all algorithms, for 100 to 2000 Accelerometer Inputs. Thus we got the context switches occurring per second for all the algorithms.

4.4.5. PAGE FAULTS PER SECOND (PFS):

Linux based utility has been used to find out the number of Page Faults occurring per Second for a program. The procedure is same as that used for context switches occurring per second. The commands used are:

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In above example abc is the executable file. A lot of information is generated using the above commands out of which we have extracted the Page faults occurring per second. Repeating the same procedure for all algorithms, we get results for all algorithms.

4.4.6. CPU USAGE:

We have used Linux based utility to find out CPU usage by a program. The procedure used is same as that for context switches occurring per second.

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In above example abc is the executable file. From the information generated, we use the CPU usage values.

4.4.7. CPU TIME:

Linux based utility has been used. The procedure is same as that for Context switches per second. The commands written on terminal are:

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In above example abc is the executable file. From the information obtained, we extract the total CPU time consumed by the program. We did the same for all algorithms and for 100 to 2000 Accelerometer Input.

4.4.8. CACHE MISS RATE:

Linux based utility has been used to find out the Cache Miss Rate (CMR) for a program. For this valgrind linux based package has to be installed. Then we have use some commands to get information about Cache Miss Rate (CMR) as given below:

```
#valgrind –tool=cachegrind “source code file name with extension”
# valgrind –tool=cachegrind ./abc
```

** In above example abc is the executable file. This gives us information about Cache Miss Rate (CMR) for all type of caches like I1, LLi, D1, LLD and LL. We did the same for all algorithms and for 100 to 2000 Accelerometer Input to get values for all algorithms.

4.4.9. READ OPERATIONS PER SECOND:

We have used Linux based utility to find out number of Read Operations occur per Second for a program. The procedure used is the same as that for Context switches occurring per second. The terminal commands are given below.

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In above example abc is the executable file. From the information we get, we mainly focus on Read Operations occurring per second for a program. We did the same for all algorithms and for 100 to 2000 Accelerometer Input to get values for all algorithms.

4.4.10. WRITE OPERATIONS PER SECOND:

Linux based utility has been used to find out the number of Write Operations occurring per Second for a program. The procedure used is same as is in context switch occurring per seconds. Finally running the following commands on the terminal give us the Write operations per second:

```
#make
# sudo ./health-check -d 60 -c -f -p “execution file name of program”
# sudo ./health-check -d 60 -c -f -p ./abc
```

Here abc is the executable file. Obtaining the number of write operations per second for one algorithm, we repeated the same for all algorithms with varying accelerometer inputs.

4.4.11. POWER CONSUMPTION:

Linux based utilities are used to find out how much power is required by a program. We tried to use the packages introduced by Colin King, namely power-calibrate and health-check on the Raspberry Pi. [28]. But unfortunately the power-calibrate package did not work out on the Raspberry pi. The power-calibrate package actually requires Direct Current (DC) supply. It measures the amount of battery power a program consumes. All possible efforts had been tried to give Direct Current (DC) supply to the Raspberry pi. When using the portable USB bank to provide the DC supply, we observed the Raspberry pi operating nicely but the power-calibrate package still not working. Thus all these steps to add external DC source went in vain.

So another way was devised to help run the power calibrate package. We used Ubuntu to calculate the power consumption. The laptop battery was used to give Direct Current (DC) and thus facilitate the working of power calibrate package. We have used reference from git clone [30] to install this package on Ubuntu. After all this the following commands were run on the terminal.

```
#make
#Sudo ./power-calibrate -c –C
```

**In above example -c flag calibrates how much power is consumed to use just 1% of the CPU and -C flag calibrates how much power is consumed in carrying out 1 context switch, on the machine. The package works only when the machine is running in Direct Current (DC).

Along with this we have to install another package name health-check package. The remaining procedure is the same as in case if calculating context switches occurring per second. The terminal commands are:

```
#make
# sudo ./health-check -d 60 -c -f -p "execution file name of program"
# sudo ./health-check -d 60 -c -f -p ./abc
```

** In above example abc is the executable file. This gives us lot if information's, but we focused on Write Operations occur per second for a program.

After all this, we get power consumed in doing 1 context switch and power consumed to use just 1% of the CPU. After running the health-check package, we obtained the number of Context Switches occurring per second and total CPU usage. After simple multiplications, power consumption for a program is obtained. We did the same for all algorithms and for 100 to 2000 Accelerometer Input. Thus the power is obtained for all algorithms using the same way for all.

CHAPTER 5: TESTING AND RESULTS

Assumption:

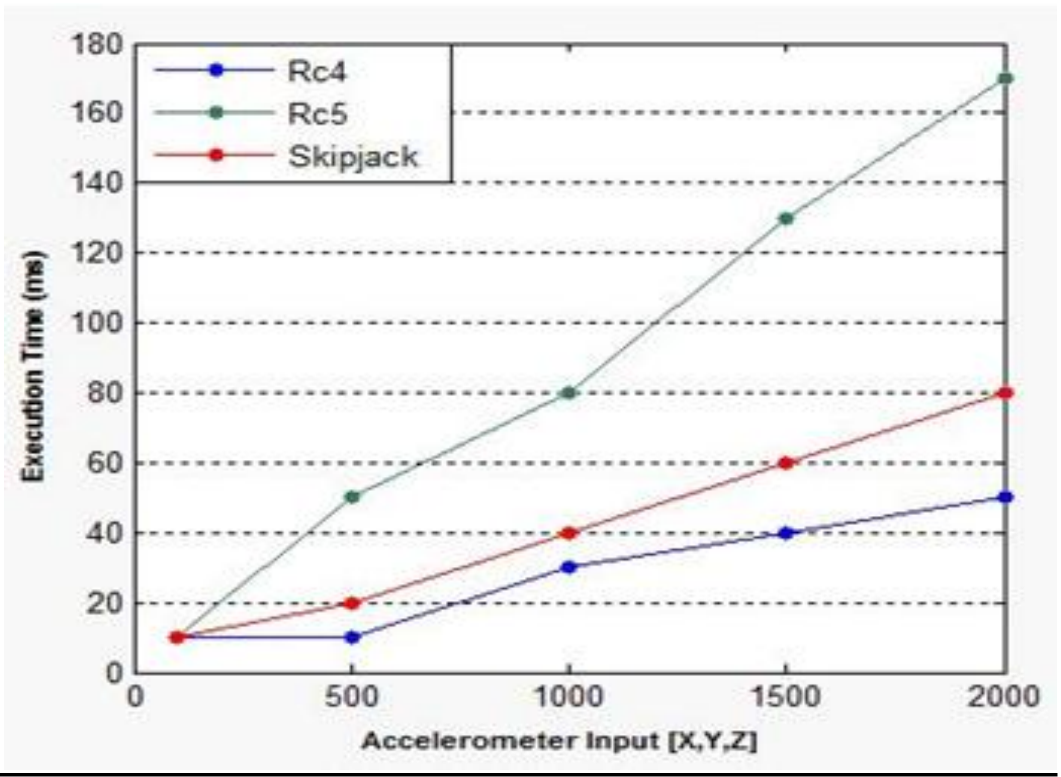
The Accelerometer readings vary from 100 to 2000 Input entries. Each individual input of the Accelerometer is a depiction of the position of a person (in 3D co-ordinate system).

5.1. EXECUTION TIME

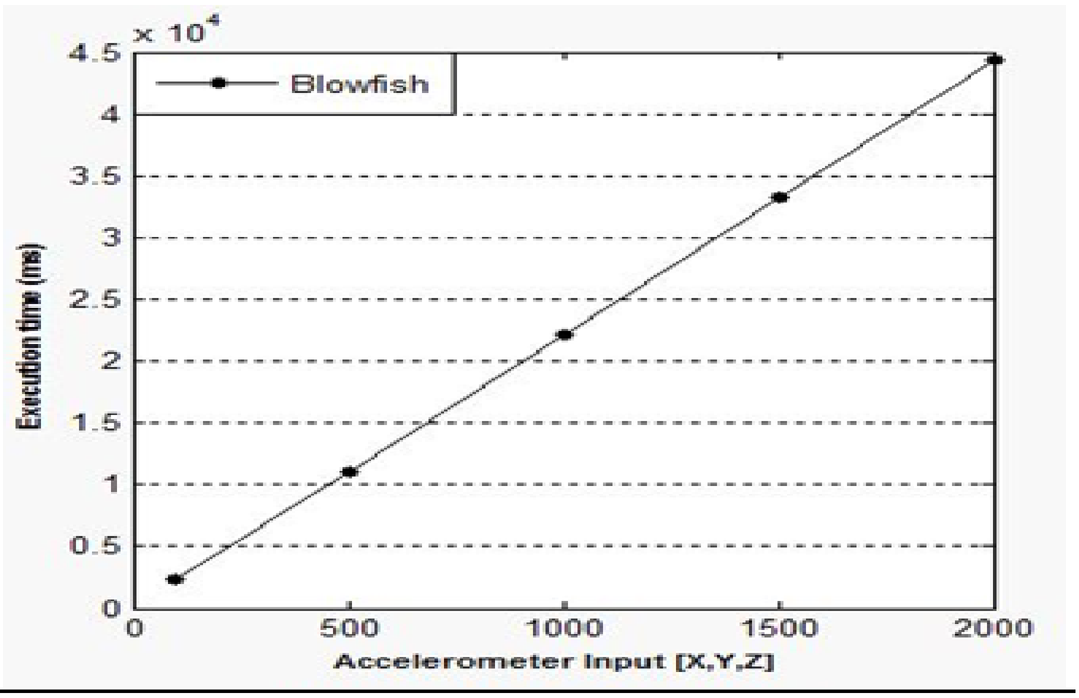
Table 5.1: Execution time for various algorithms (in ms) on variation of accelerometer readings

Algorithm	Accelerometer Readings(AR)				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Rc4	10	10	30	40	50
Rc5	10	50	80	130	170
Skipjack	10	20	40	60	80
Blowfish	2280	11100	22200	33280	44420

**Execution time (in ms)



Graph 5.1: Execution times verses Accelerometer input for RC4, RC5 and Skipjack

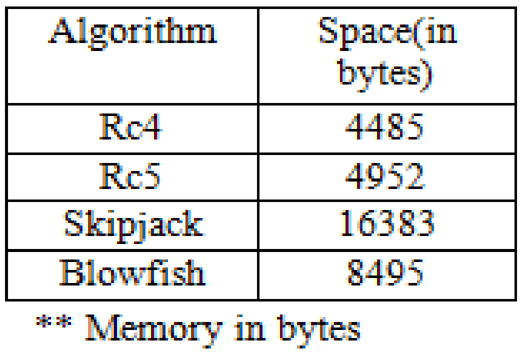


Graph 5.2: Execution times verses Accelerometer input for Blowfish

As is eminent from the graph 5.1 and 5.2, Blowfish takes a time that is far more than the rest of the algorithms. Also RC4 algorithm requires the minimum time as compared to the rest (for all sizes of Input data.) In Body Sensor Network (BSN), where the time consumption is a major constraint, we find RC4 to be the best suitable choice.

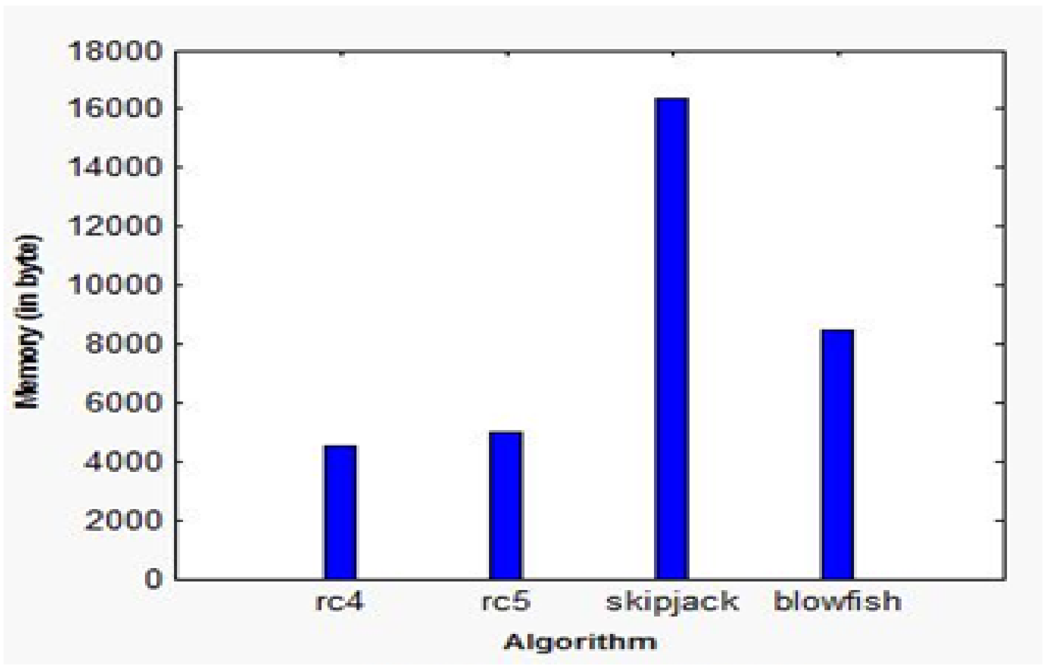
5.2. MEMORY OCCUPIED AT RUNTIME

Table 5.2: Memory occupied by various algorithms (in bytes)



Algorithm	Space(in bytes)
Rc4	4485
Rc5	4952
Skipjack	16383
Blowfish	8495

** Memory in bytes



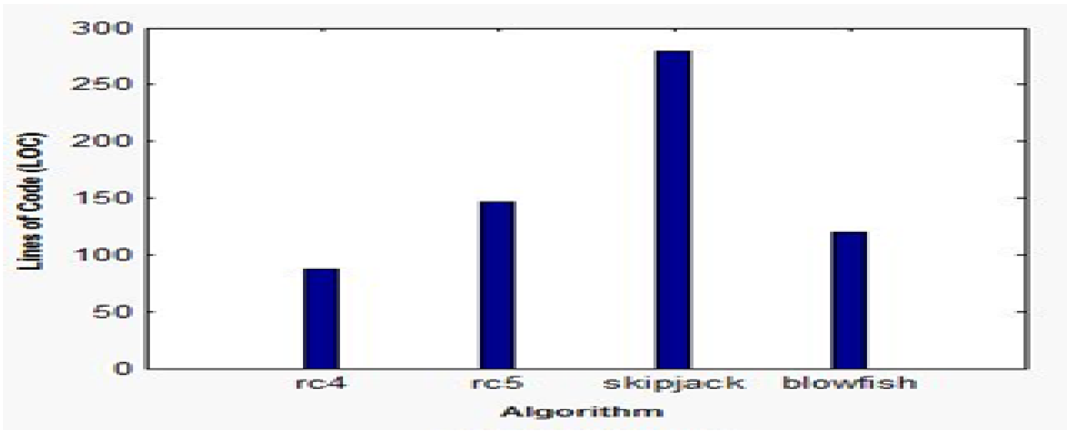
Graph 5.3: Memory occupied verses algorithm

As is seen from the graph 5.3, Skipjack algorithm uses a larger amount of memory space as compared to the rest of the algorithms; whereas RC4 occupies minimum memory at run time. In BSN where memory management is necessary, we can say RC4 is a good choice.

5.3. LOC (LINES OF CODE)

Table 5.3: Lines of Code (LOC) for various algorithms

Algorithm	LOC
Rc4	88
Rc5	147
Skipjack	279
Blowfish	120



Graph 5.4: Lines of code verses algorithm

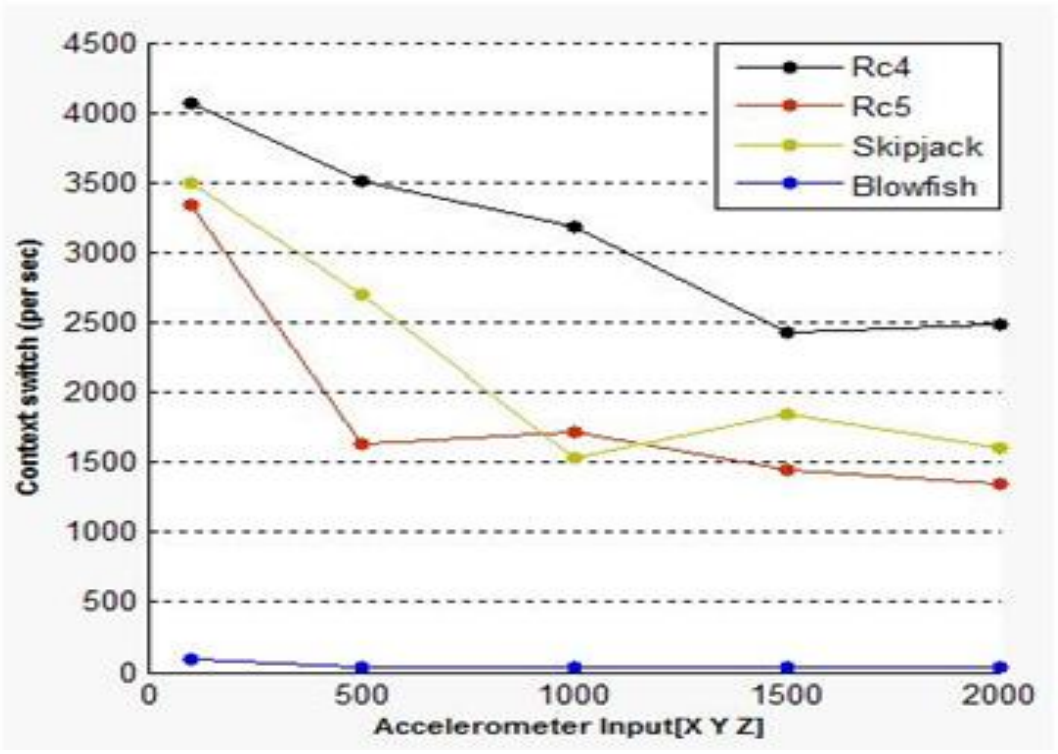
As graph 5.4 shows, Skipjack has maximum Lines of Code (LOC) than all the other Algorithms and RC4 has minimum Lines of Code (LOC). Since BSN have less storage space, less number of Lines of Code (LOC) is preferred, which justifies RC4 to be a good choice.

5.4. CONTEXT SWITCHES PER SECOND

Table 5.4: Context switches for various algorithms on variation of accelerometer readings

	Context switches (per sec)				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Algorithm	Total/sec	Total/sec	Total/sec	Total/sec	Total/sec
Rc4	4063.16	3508.61	3184.29	2429.44	2487.23
Rc5	3344.8	1630.95	1714.68	1444.27	1338.79
Skipjack	3494.94	2697.81	1537.81	1840.47	1600.18
Blowfish	86.32	39.87	35.41	31.67	30.62

**Context switches per sec.



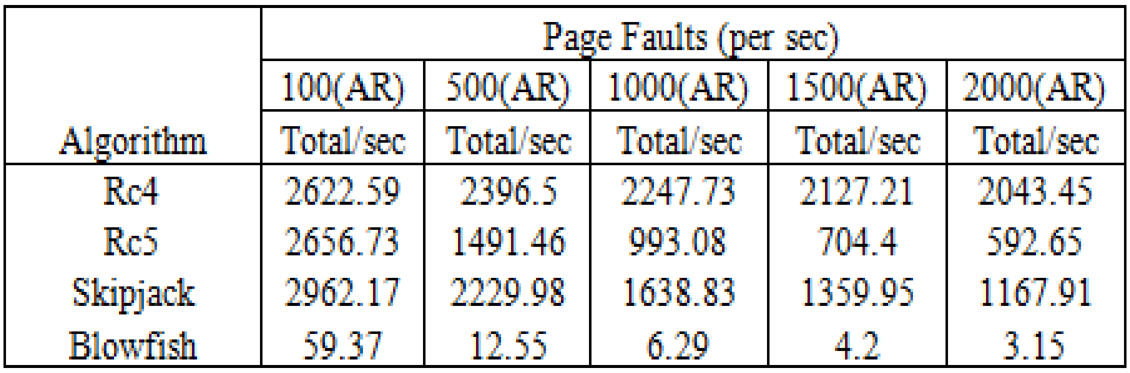
Graph 5.5: Context switches per sec verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

We conclude from the graph 5.5 that maximum Context Switches per Second (CSS) occur for RC4 algorithm whereas Minimum Context Switches per Second (CSS) occurs for Blowfish.

Thus the performance of Blowfish algorithm in terms of Context Switches per Second (CSS) is better than all other algorithms. So in Body Sensor Network (BSN), we can say that Blowfish algorithm is a good choice.

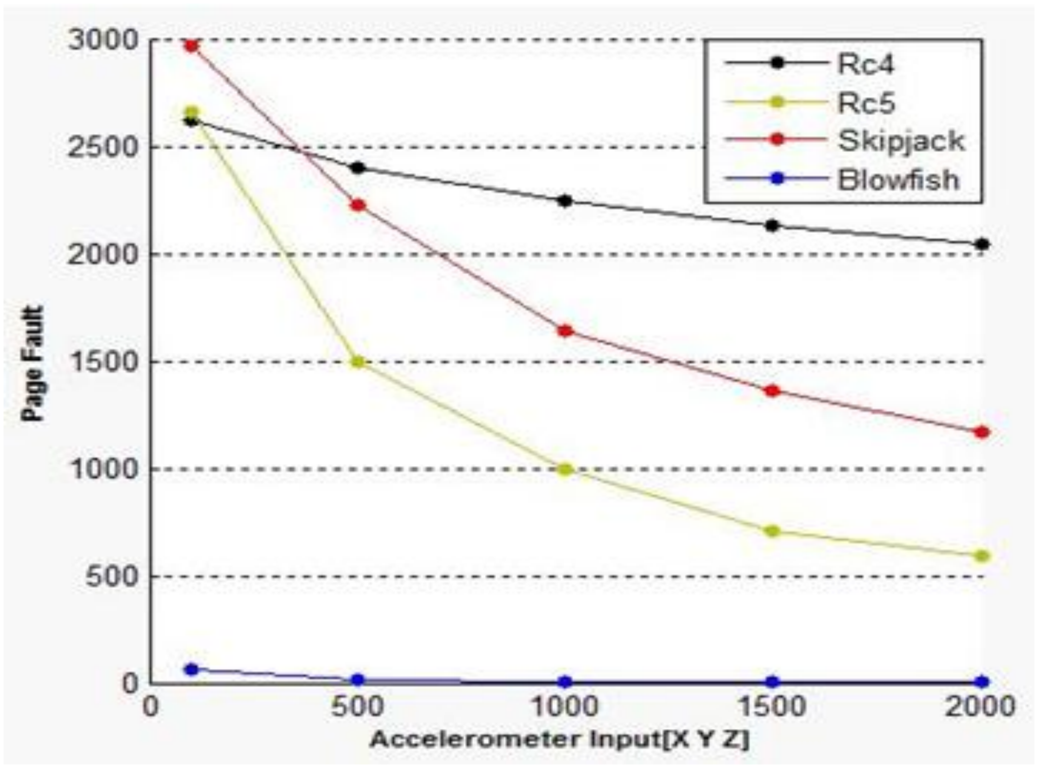
5.5. PAGE FAULTS PER SECOND

Table 5.5: Page faults per second for various algorithms on variation of accelerometer readings



	Page Faults (per sec)				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Algorithm	Total/sec	Total/sec	Total/sec	Total/sec	Total/sec
Rc4	2622.59	2396.5	2247.73	2127.21	2043.45
Rc5	2656.73	1491.46	993.08	704.4	592.65
Skipjack	2962.17	2229.98	1638.83	1359.95	1167.91
Blowfish	59.37	12.55	6.29	4.2	3.15

** Page faults per sec.

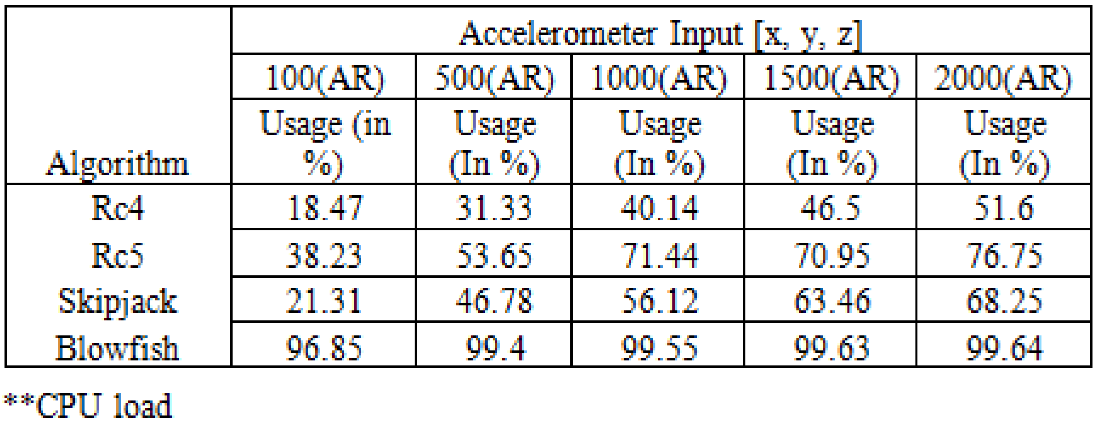


Graph 5.6: Page faults verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

As we can see in above table 5.5 and graph 5.6, minimum Page Faults per Second (PFS) occur for Blowfish algorithm for all sizes of input and maximum Page Faults per Second (PFS) occur for skipjack algorithm. But as value of inputs increase, we find the maximum Page Faults per Second (PFS) occurring in case of RC4 algorithm. Thus, as far as Page Faults per Second (PFS) are concerned, we conclude that Blowfish Algorithm is the best suitable algorithm. RC4 is not suitable when the amount of input data becomes large.

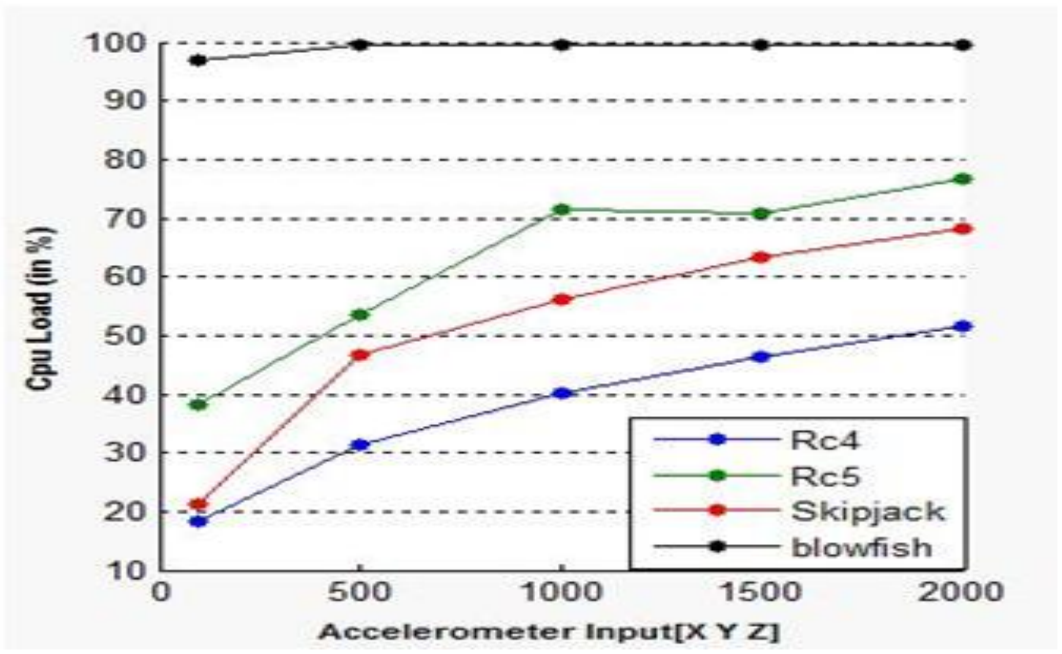
5.6. CPU USAGE

Table 5.6: CPU Usage for various algorithms on variation of accelerometer readings



Algorithm	Accelerometer Input [x, y, z]				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
	Usage (in %)	Usage (In %)	Usage (In %)	Usage (In %)	Usage (In %)
Rc4	18.47	31.33	40.14	46.5	51.6
Rc5	38.23	53.65	71.44	70.95	76.75
Skipjack	21.31	46.78	56.12	63.46	68.25
Blowfish	96.85	99.4	99.55	99.63	99.64

**CPU load

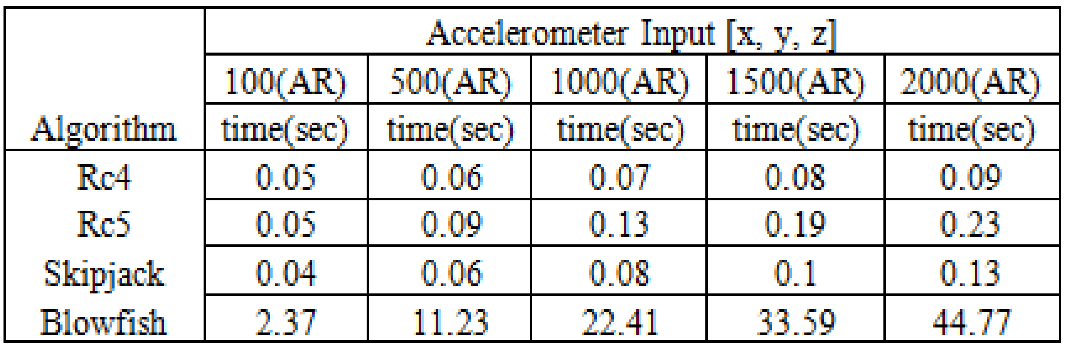


Graph 5.7: CPU Load verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

We observe that the maximum load on CPU is incurred in case of Blowfish algorithm and the minimum load is for RC4 algorithm for all sizes of input data. Hence, the performance of RC4 Algorithm in terms of CPU load is better than all other algorithms. In all the cases where CPU load is a constraint, we conclude the use of RC4 algorithm to be the best.

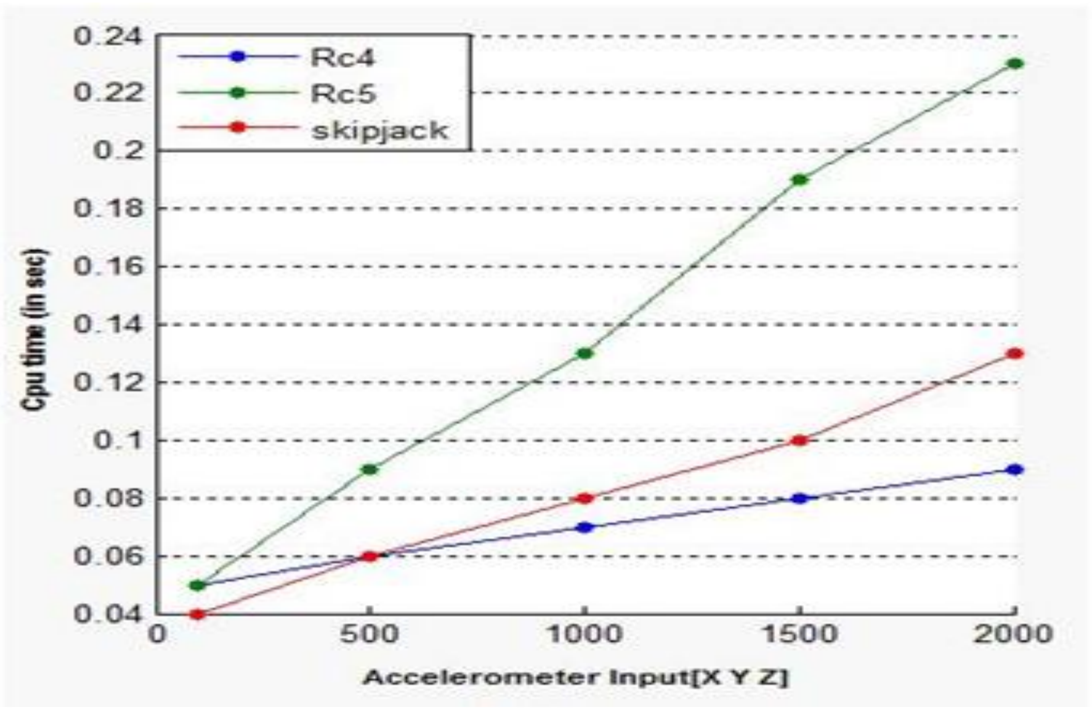
5.7. CPU TIME

Table 5.7: CPU time for various algorithms (in ms) on variation of accelerometer readings

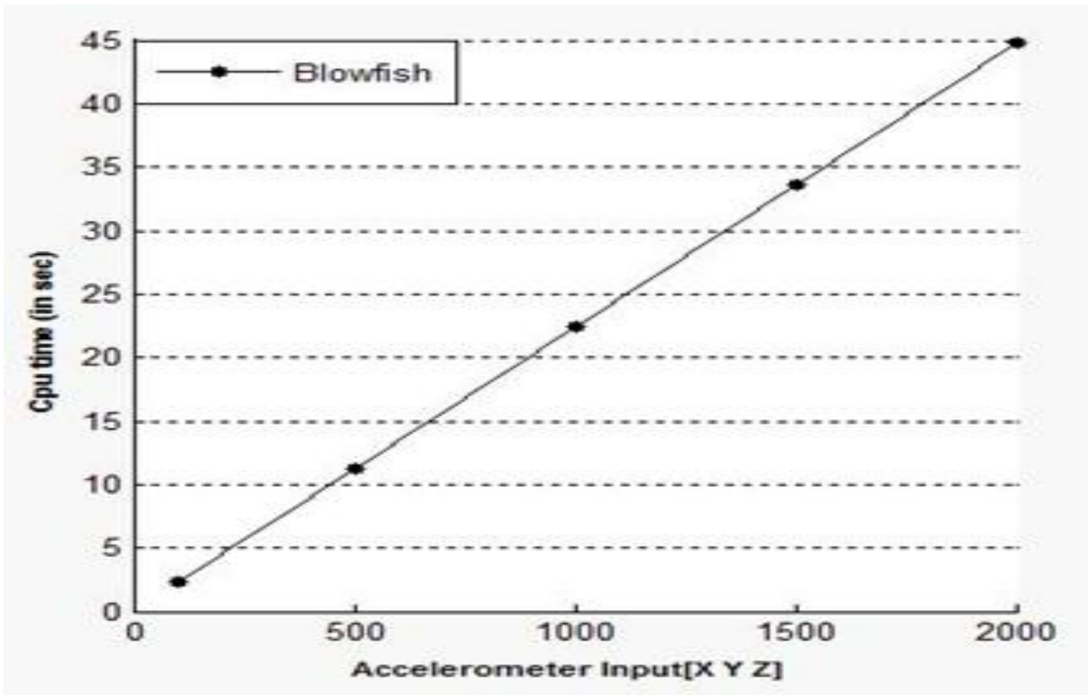


	Accelerometer Input [x, y, z]				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Algorithm	time(sec)	time(sec)	time(sec)	time(sec)	time(sec)
Rc4	0.05	0.06	0.07	0.08	0.09
Rc5	0.05	0.09	0.13	0.19	0.23
Skipjack	0.04	0.06	0.08	0.1	0.13
Blowfish	2.37	11.23	22.41	33.59	44.77

** CPU time



Graph 5.8: CPU time verses Accelerometer input for RC4, RC5 and Skipjack

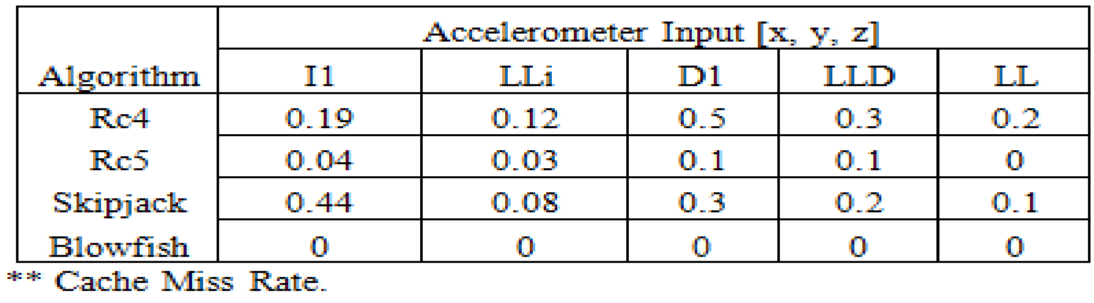


Graph 5.9: CPU time verses Accelerometer input for Blowfish

As is seen from the graphs 5.8 and 5.9, Blowfish Algorithm takes maximum CPU time. We also observe Skipjack algorithm taking minimum CPU time for small sizes of Input data. If the size of input is increased, the performance of RC4 gets better. We thus conclude that at all places where CPU time is a constraint, the use of algorithm can be decided depending on amount of data. If we work for small amounts of data, Skipjack is a good choice but as the amount of data keeps increasing, the performance of RC4 keeps getting better and it poses a better alternative. Blowfish in all cases is not considered a good choice.

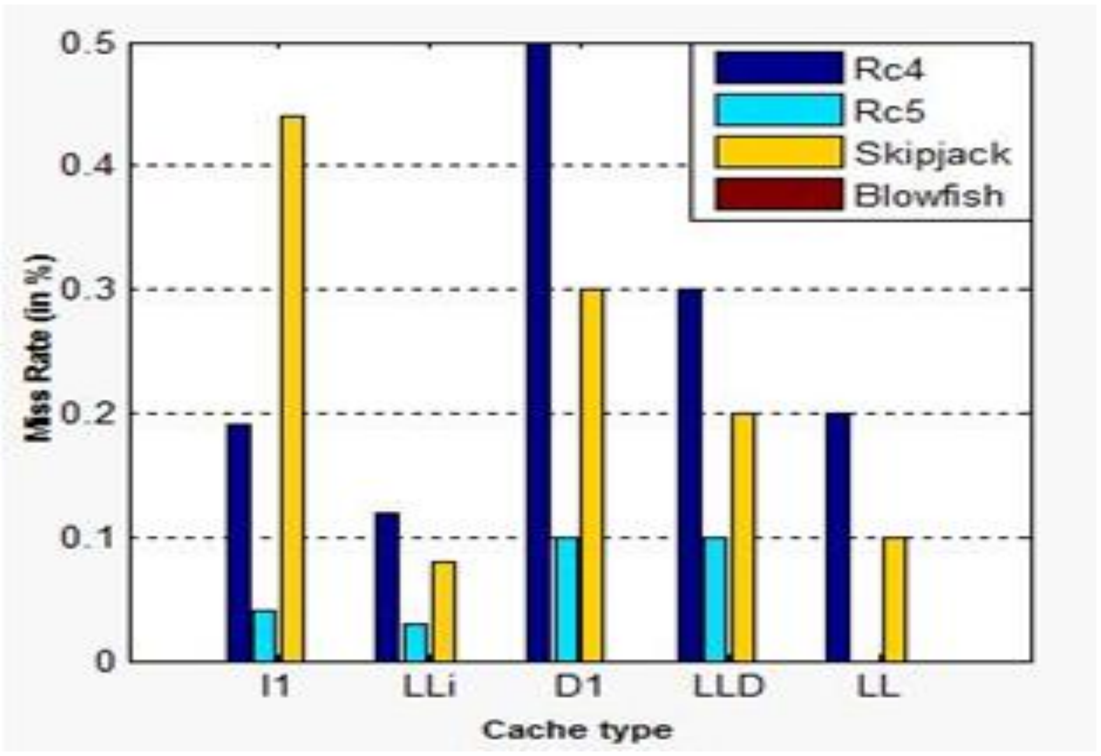
5.8. CACHE MISS RATE

Table 5.8: Cache miss rate for various algorithms (in ms) on variation of accelerometer readings



Algorithm	Accelerometer Input [x, y, z]				
	I1	LLi	D1	LLD	LL
Rc4	0.19	0.12	0.5	0.3	0.2
Rc5	0.04	0.03	0.1	0.1	0
Skipjack	0.44	0.08	0.3	0.2	0.1
Blowfish	0	0	0	0	0

** Cache Miss Rate.



Graph 5.10: Cache miss rate verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

Here we focus on all the different kinds of caches. Considering the statistics of I1 cache, we conclude, the maximum Cache Miss Rate (CMR) to be for Skipjack algorithm and minimum for Blowfish Algorithm which is closely followed by RC5 Algorithm. For the case of LLi cache, we observe the maximum Cache Miss Rate (CMR) for RC4 Algorithm and minimum for Blowfish Algorithm which is succeeded by RC5 Algorithm. For D1 cache, the maximum Cache Miss Rate (CMR) is for RC4 Algorithm and minimum for Blowfish Algorithm after which is RC5 algorithm. If we consider LLD cache, we find the maximum Cache Miss Rate (CMR) to be for RC4 Algorithm and minimum for Blowfish Algorithm after which is the RC5 Algorithm. For the statistics of LL cache, the maximum Cache Miss Rate (CMR) is for RC4 Algorithm and minimum is for Blowfish and RC5 Algorithm which is followed by RC4 Algorithm.

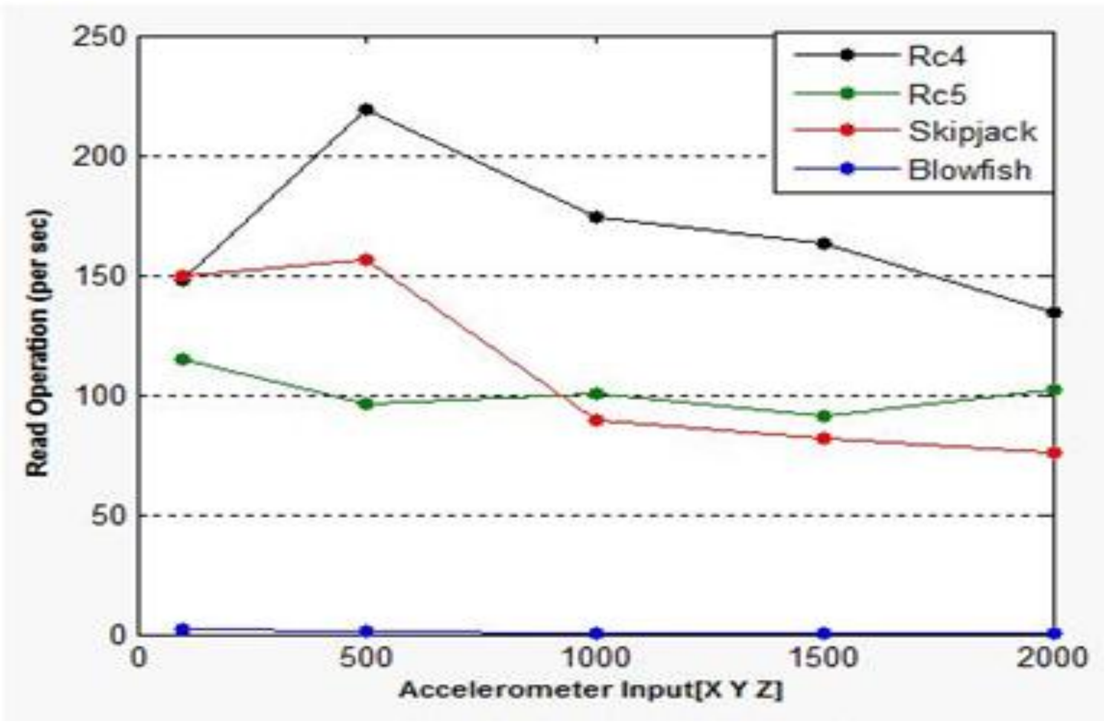
Hence we reach the conclusion; the Cache Miss Rate (CMR) for Blowfish Algorithm is 0 percent for all type of caches. In terms of Cache Miss Rate (CMR), the worst choice for I1 cache is skipjack Algorithm, for LLi cache is RC4 Algorithm, for D1 cache is RC4 Algorithm, for LLD cache is RC4 Algorithm, and for LL cache is RC4 Algorithm. In all cases where the cache miss rate (CMR) is a major constraint, the best choice considering all types of caches is Blowfish algorithm followed by RC5.

5.9. READ OPERATIONS PER SECOND

Table 5.9: Read operations per second for various algorithms (in ms) on variation of accelerometer readings

	Read Operations (per sec)				
Algorithm	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Rc4	147.75	219.29	173.93	162.74	134.17
Rc5	114.68	96.57	100.02	91.22	102.33
Skipjack	149.17	155.94	89.8	81.6	75.84
Blowfish	2.53	0.89	0.58	0.51	0.49

**Read Operations per second.

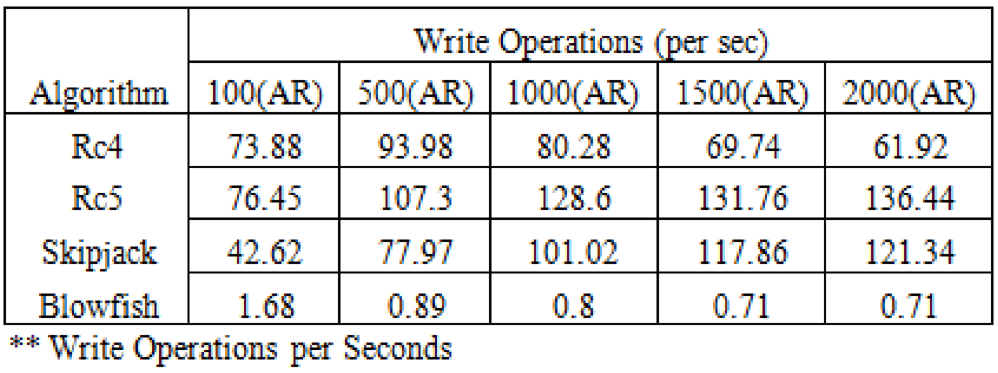


Graph 5.11: Read operations per second verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

Considering the Read Operations per Second (ROS) vs. Accelerometer-Input, we find the maximum Read Operations per Second (ROS) to be for RC4 Algorithm and minimum for Blowfish Algorithm for any size of Accelerometer-Input. Also greater the number of Read operations per second easier it is for processor to Read the whole content in lesser time. In BSN where Read Operations per Second (ROS) is major issue, the best choice in terms of Read Operations per Second (ROS) is RC4 algorithm and worst choice is Blowfish algorithm for all sizes of Accelerometer-Input.

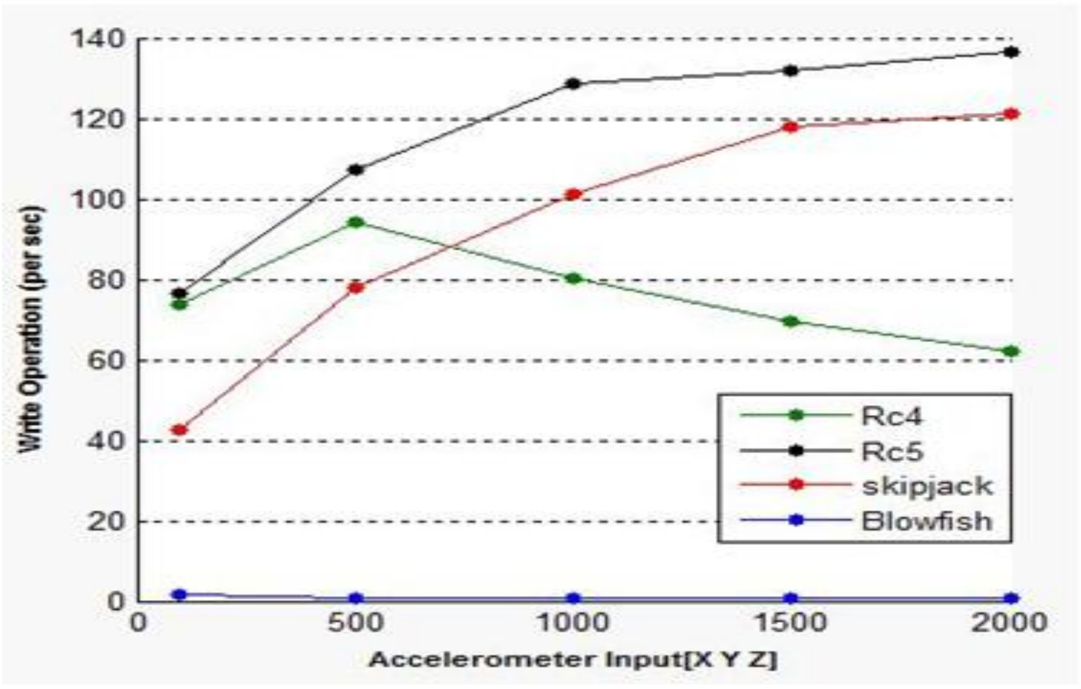
5.10. WRITE OPERATIONS PER SECOND

Table 5.10: Write operations per second for various algorithms (in ms) on variation of accelerometer readings



	Write Operations (per sec)				
Algorithm	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
Rc4	73.88	93.98	80.28	69.74	61.92
Rc5	76.45	107.3	128.6	131.76	136.44
Skipjack	42.62	77.97	101.02	117.86	121.34
Blowfish	1.68	0.89	0.8	0.71	0.71

** Write Operations per Seconds



Graph 5.12: Write operations per second verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

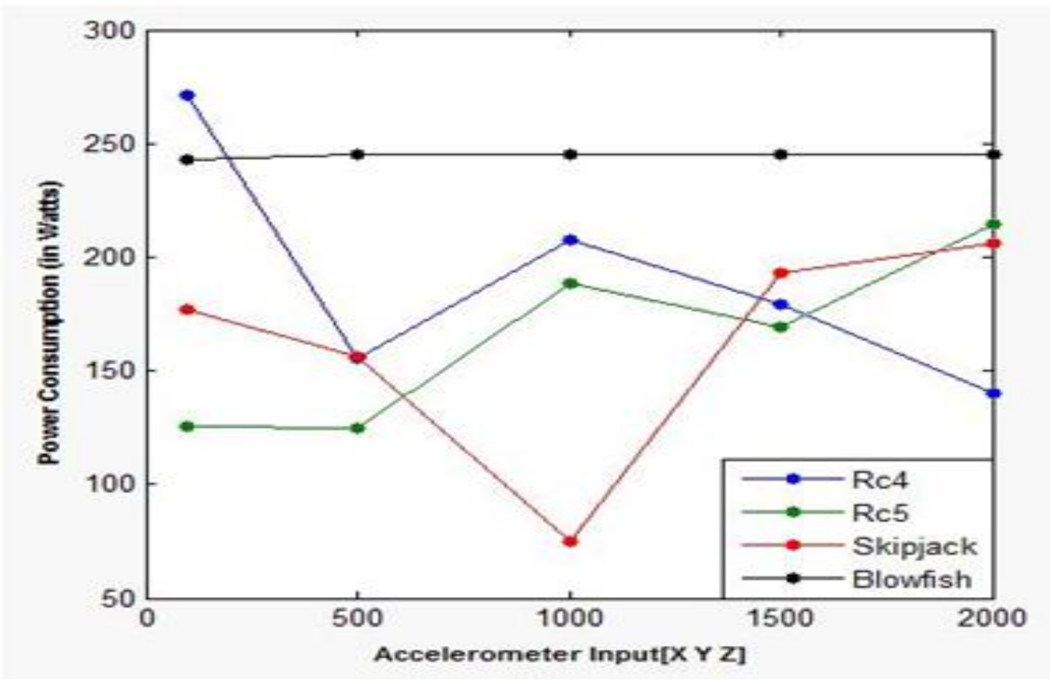
Focusing on the statistics for Write Operations per Second (WOS) vs. Accelerometer-Input, we find the maximum Write Operations per Second (WOS) to be for RC5 algorithm and minimum for Blowfish algorithm for any sizes of Accelerometer-Input. Also greater the number of write operations occurring per second, easier it is for processor to write the whole content in less time. In cases where Write Operations per Second (WOS) is major constraint, we conclude the best choice to be RC5 algorithm and worst to be Blowfish algorithm for all sizes of Accelerometer-Input.

5.11. POWER CONSUMPTION

Table 5.11: Power consumption for various algorithms (in watts) on variation of accelerometer readings

Algorithm	Accelerometer Input [x , y, z]				
	100(AR)	500(AR)	1000(AR)	1500(AR)	2000(AR)
	power(watts)	power(watts)	power(watts)	power(watts)	power(watts)
Rc4	271.37	155.41	207.23	178.95	140.15
Rc5	125.53	124.91	188.37	169.59	214.84
Skipjack	176.69	155.96	75.15	193.4	205.89
Blowfish	243.14	245.32	244.82	245.51	245.55

**Power Consumption (in Watts)



Graph 5.13: Power consumption verses Accelerometer input for RC4, RC5, Skipjack and Blowfish

Laying a focus on the statistics for Power Consumption (PC) vs. Accelerometer-Input, we observe the maximum power consumed to be by Blowfish Algorithm for any size of Accelerometer-Input. Also RC5 algorithm consumes minimum power for almost every Accelerometer-Input, except when the input is 1000 and 2000. This is because with increase in the size of input, the number of context switches per second decreases and the CPU load increases. For the input case of 1000, the CPU load is minimum and Skipjack Algorithm consumes minimum power. Further increasing the input, the CPU load increases as usual and thus consumes extra power. When the Accelerometer-Input is 2000, the power consumed by RC4 Algorithm, is minimum. Also an important factor to consider is that lesser the power, an Algorithm consumes, the more suitable it becomes for Small Board Computers (SBC). In BSN, where Power Consumption (PC) is a major constraint as BSNs are low power devices, the best choice as far as Power Consumption is concerned is RC5 algorithm for small size of input data and RC4 algorithm for large sizes of input data. The worst choice in terms of Power Consumption (PC) is Blowfish algorithm for all sizes of Accelerometer input.

CHAPTER 6: ISSUES WITH BSN ALGORITHMS

6.1. SPECIAL CASE OF SENSOR NODES

Security in sensor devices is different from normal cases as sensor devices are special in a variety of ways. Also implementing security of data for these devices should be done keeping in mind the type of device and its resource and computational limitations.

i) The fact that separates sensor node data encryption from normal ones is its limited battery. The sensor node battery cannot be recharged and hence for encrypting data, if the sensor nodes remain open for a long time, the battery gets fully discharged and will not be able send any signal further to any other node. Thus such an optimization is a challenging problem.

ii) Also since data to be sent over the sensor device is small a very large block size is not desirables.

iii) The sensor devices have a low memory and thus even need a memory efficient algorithm [11].

iv) Transmission of data is one of the most energy consuming tasks undergone by a node - using data compression to reduce the number of bits sent reduces energy expended for transmission. Data compression which highly reduces the communication overhead by aggregating and compressing data packets is performed at intermediate sensor nodes [12].

6.2. ISSUES WITH THE ENCRYPTION ALGORITHMS

I) The symmetric key algorithms such as DES etc can be used but the major drawback they face is that their key needs to be secret.

II) Also Symmetric encryption algorithms seem to be inherently well suited to low-end devices, because they offer a relatively low overhead [2].

III) For public key encryption techniques used, we have to do massive computation for encrypting any plain text. Also sometimes these methods may not be also suitable sensor networks [11].

IV) **RC-4**

i) It is considered to be efficient if key length is greater than 128 bits [24].

ii) Also the implementation of RC4 in WEP (Wired equivalent privacy), to secure wireless sensor networks, is not considered to be too efficient. The problem there is not the RC 4 algorithm but the way in which it is used [24].

iii) RC4 also is not able to match the standards set by cryptographers for a secure cipher in various ways, and thus is not recommended for use in new applications [24].

iv) Unlike a modern stream cipher, RC4 does not take a separate nonce along with the key. So if a single long-term key is to be used to encrypt multiple streams, the cryptosystem must specify how to combine the nonce and the long-term key to generate the stream key in case of RC4. *One way of tackling this is by generating a "new" RC4 key by hashing a long-term key with a nonce. Still many applications using RC4 concatenate the key and nonce and RC4's weak key schedule then gives rise to a variety of serious problems. [24]

v) The bytes RC4 produces are not always random- they contain small biases. From the point of view of cryptography, this is not at all desired. Encrypting the same message (plaintext) with many different RC4 keys should give a new cipher text each time. But there are biases which can help the intruder to break into the cipher text [16].

vi) Because RC4 is a stream cipher, it is more prone than common block ciphers. If a strong message authentication code (MAC) is not added with it, then the cipher text is vulnerable to a bit-flipping attack. Incorrect implementation can also lead to a stream cipher attack. Furthermore, it can happen that a double encryption of the message with the same key may output the plaintext again rather than cipher text because of the fact that the XOR function would result in the second operation reversing the first [17].

vii) In 2013, a new attack was proposed by AlFardan, Bernstein, Paterson, Poettering and Schuldt that could use new statistical biases in RC4 key table to recover plaintext with large number of TLS encryptions [17].

V) BLOWFISH

i) Blowfish is known to be susceptible to attacks on weak keys [27].

ii) Blowfish is one of the fastest block ciphers; the problem arises only when changing keys. Each new key requires a pre-processing to be done which is equivalent to encrypting about 4 kilobytes of text, this is comparatively very slow .This prevents its use in certain applications, but is not a problem in others, such as Splash ID [27].

iii) The problem also arises as it must give a key to the person involved in transmission specifically not through the unsecured transmission channel. Each pair of users needs a unique key, so as the numbers of users are increased, key management becomes more and more complicated. For example N*(N-1)/2 keys required. It also has a weakness in the decryption process over other algorithms in terms of the time consumption and serially in throughput [27].

VI) **RC-5**

i) In the case of a sensor network, the costs of call setup and return outweigh the costs of the RC5 itself [1].

ii) RC5 is word-oriented. In comparison with RC4, RC5 consumes more code memory size [1].

iii) It also needs a pre-computed key schedule to be stored in memory, which leads to occupation of significant bytes of memory for each key [1].

iv) Even though the RC5 algorithm can be small, the common RC5 libraries are too large to fit on a platform [1].

VII) **SKIPJACK**

i) Its susceptibility to shortcut attacks, wherein the intruder can exploit some property of the encryption algorithm from which the key or plaintext can be determined in much less time than by exhaustive search, leads to a challenge in its use today [18].

ii) The key length is longer making brute force attacks millions of times slower; however 80 bits can also be prone to brute force attacks [19].

6.3. SITUATION BASED USE

A. RC-4

i) Rc4 is extremely efficient in software implementations since only byte operations are used [24].

ii) It can be considered secure if keys of length higher than 128 bits are used [24].

iii) In WEP, RC4 in combination with a particular method for generating its keys was broken [24].

iv) If RC4 is used (i.e. any stream cipher technique) instead of DES (i.e. block cipher technique) in the output feedback mode, the net encryption and decryption time decreases at sensor node [24].

v) RC4 is an extremely popular cipher for SSL/TLS connections. There are two main reasons for it namely -RC4 does not need a padding or IV (Initialization vectors), which implies it's immune to recent TLS attacks like BEAST and Lucky13. Also RC4 is very fast. Thus a fast encryption implies a less computation and therefore lower hardware requirements which are beneficial for service providers like Google [25].

B. BLOWFISH

i) Blowfish is considered to be suitable for wireless network application which exchange packets of comparatively small sizes [25].

ii) The password-hashing method used in Open BSD uses an algorithm derived from Blowfish which uses the slow key schedule; the basic purpose of using it is that the extra computational effort required provides the much needed protection against dictionary attacks [27].

iii) Also Blowfish is free for use to anyone. This helps increase its usage and popularity in today’s time [27].

iv) Blowfish is said to be efficient in software, at least on some software platforms (it uses key-dependent lookup table; hence the performance depends on how the platform will handle memory and caches) [32].

C. RC-5

i) RC5 is patented by RSA and can be used as a replacement for DES with block size=64 bit and key size =56 bits [26].

ii) It is used in IPSec Encapsulating Security Payload (ESP).It is used here in the CBC mode [20].

iii) It is even a good potential for use in Wireless Body sensor networks [2].

iv) It is used to provide security connections in LSWN (Low speed wireless networks) [31].

D. SKIPJACK

i) The algorithm was developed to be used in voice, fax and secure telephones, such as the AT&T TSD-3600. It was also used in the first Fortezza Crypto Card. (By US Govt) [21].

ii) Skipjack is said to be immune to exhaustive search for time to come till the time we don't have further improvements in the exhaustive search techniques [22].

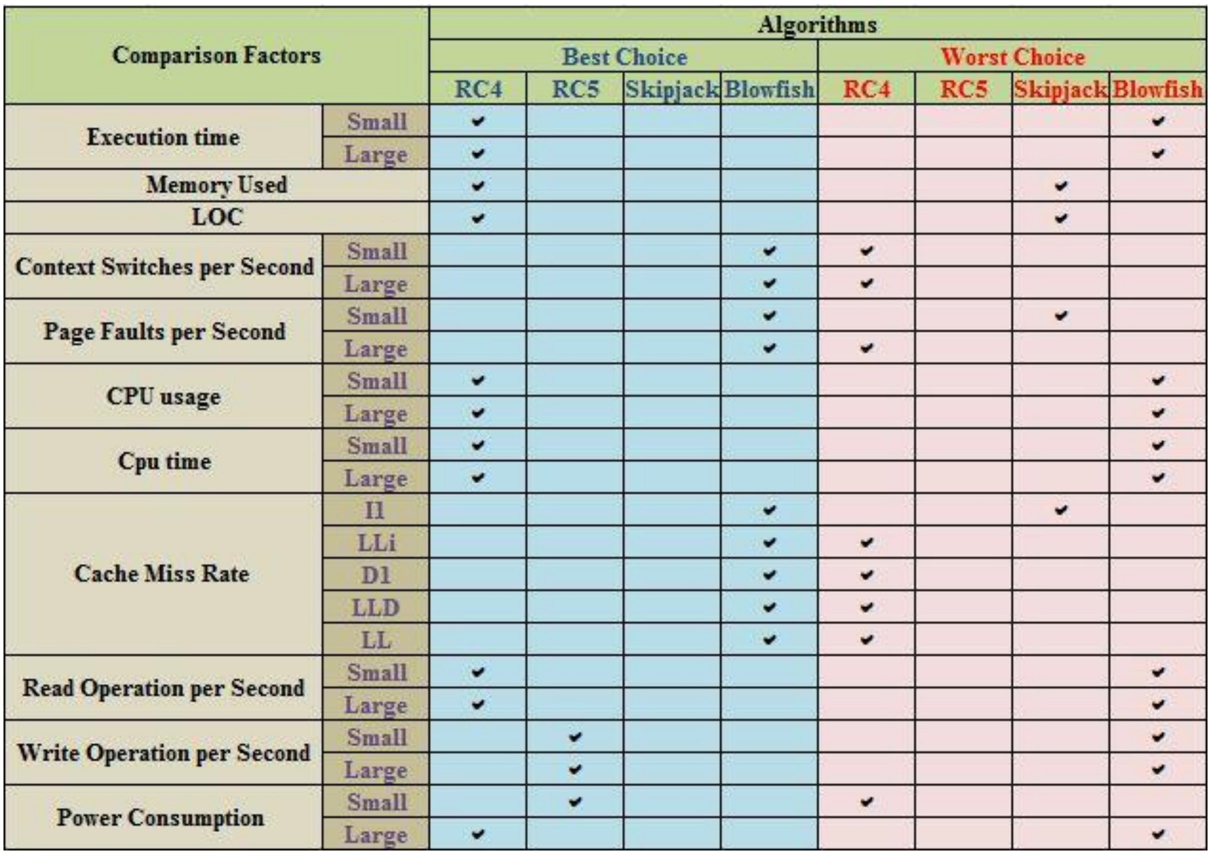
iii) Also the key of the Skipjack algorithm offers a greater margin of security than single encryption DES, but this margin is comparatively small and is overcome by the purely economic and technical considerations. Also an important factor to consider is that the increasing key size in Skipjack does not necessarily increase costs as in single encryption DES [22].

iv) It can also be used for data encryption in computer networks (Defense dept uses it in defense messaging system.) [23].

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

7.1. CONCLUSIONS

Table 7.1: A complete summation table of all the results obtained.



Comparison Factors		Algorithms							
		Best Choice				Worst Choice			
		RC4	RC5	Skipjack	Blowfish	RC4	RC5	Skipjack	Blowfish
Execution time	Small	✓							✓
	Large	✓							✓
Memory Used		✓						✓	
LOC		✓						✓	
Context Switches per Second	Small				✓	✓			
	Large				✓	✓			
Page Faults per Second	Small				✓			✓	
	Large				✓	✓			
CPU usage	Small	✓							✓
	Large	✓							✓
Cpu time	Small	✓							✓
	Large	✓							✓
Cache Miss Rate	I1				✓			✓	
	LLi				✓	✓			
	D1				✓	✓			
	LLD				✓	✓			
	LL				✓	✓			
Read Operation per Second	Small	✓							✓
	Large	✓							✓
Write Operation per Second	Small		✓						✓
	Large		✓						✓
Power Consumption	Small		✓			✓			
	Large	✓							✓

The best and worst choice of algorithms has been depicted depending on the comparison factor considered. Small and large indicate the inputs of accelerometer (which algorithm is best and worst for small accelerometer reading and which for large). In the Cache miss rate, a distinction has been shown for various types of caches and the best and worst algorithm for cache miss rate of a particular type of cache.

Secure and efficient encryption algorithms are very important for the proper working of BSN. Various factors need to be addressed depending on the situation. In the paper, we first reviewed the common algorithms used for providing a secure data transmission and then selected symmetric key cryptography algorithms for the comparison task. Symmetric key algorithms were used as they offered comparatively less computation.

As per the experiments conducted, various conclusions were drawn about the performance of the algorithms (Refer Table 7.1). From Table 7.1, we inferred that the execution time for Blowfish algorithm is the maximum, while it is minimum for RC 4. In BSN where execution time needs to be minimum, RC4 is the best suitable choice.

Also the LOC and memory occupation of RC4 is minimum which is beneficial in case we have a memory limitation.

For other cases when we consider the context switches per second or the page faults occurring per second, we get Blowfish as a better option than others. All the places where load on CPU is a constraint, CPU usage and CPU time are best in case of RC4. For the cache miss rate, we consider all the types of caches namely I1, LLi, D1, LLD, LL cache. The cache miss rate is 0% in all cases for Blowfish and thus increases speed to a great deal. The corresponding Read and Write operations per second have RC4 and RC5 respectively as the best algorithms.

The last factor of power consumption gives results that tell us that for small number of inputs RC5 is a good choice and as the number of inputs increase, the choice shifts from RC5 to RC4.

Further we even conclude that RC4 is efficient in software implementations and its security increases further if the key size is increased to greater than 128 bits. The biggest drawback of RC4 lies in the fact that the cipher text it produces contains biases which can prove to be detrimental to sensitive information being sent.

Blowfish algorithm is said to be a bit slower than the rest but is extremely beneficial for small packets sent over the wireless networks. It is free to use and this feature further increases its popularity.

As far as RC5 algorithm is concerned, its code memory size is greater as compared to rest of the algorithms. Also the libraries required to implement it are sometimes too large to fit on a platform. Despite all this is works decently find for wireless system security.

Skipjack algorithm is comparatively a new algorithm. Its key size being 80 bits is sometimes said to be susceptible to Brute force attacks. On the whole it is said to be immune to exhaustive search and is also for data encryption in Computer networks. Skipjack being a new algorithm is not so widely used but has a great potential to securely transmit data.

Through this work, we established a comparison among various symmetric key algorithms used in BSN. Also we have put forth a comparison which will help decide which algorithm is best suited for which kind of situation.

7.2. FUTURE WORK

As of now all the algorithms have been implemented and tested considering Raspberry pi as a prototype. The entire code for encryption and decryption works on the pi and depending on the results obtained we concluded which algorithm to be the best suited under a particular performance metric. In future, we plan to implement the encryption part on the Raspberry pi and decryption part on the android based phone. This will help us track an algorithm's performance under conditions (such as their execution time, CPU usage, Cache miss rate, Page faults, Context Switches, Read Operations per Second, Write Operations per Second, and Power Consumption etc) taking the entire system under consideration. The results obtained can be used to obtain an algorithm's performance during actual data communication. In sensor devices, where a secure transmission of data is a need of the time, the work will be extremely useful.

REFERENCES

1. Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, J. D. Tygar, "SPINS: Security Protocols for Sensor Networks", Mobile Computing and Networking, Rome, Italy, 2001.
2. Dhanashri H. Gawali and Vijay M. Wadhai, "rc5 algorithm: potential cipher solution for security in wireless body sensor networks (wbsn)", International Journal Of Advanced Smart Sensor Network Systems, July 2012.
3. Cool Pi Projects, Cool Pi Projects| IT2Pi Programming inventors and expo, Greenville SC 2013, http://www.raspi-greenville.org/cool-pi-projects/ (July 2, 2014)
4. Bernadette Johnson, How the Raspberry Pi works, (HowStuffworks-How the Raspberry pi works), http://computer.howstuffworks.com/raspberry-pi4.htm (July 2, 2014)
5. William Stallings, Cryptography and network security, third edition, Pearson, 2002.
6. Yao Minglin, TangShan Coll., Tangshan, Ma Junshuang, "Stream Ciphers on wireless sensor networks", Third International Conference on Measuring Technology and Mechatronics Automation, January 2011.
7. Shervin Amini, Richard Verhoeven, Johan Lukkien, Shudong Chen, "Toward a Security Model for a Body Sensor Platform", IEEE International Conference on Consumer Electronics, 2011.
8. Diaa Salama Abd Elminaam, Hatem Mohamed Abdual Kader, and Mohiy Mohamed Hadhoud, "Evaluating The Performance of Symmetric Encryption Algorithms", International Journal of Network Security, May 2010.
9. Antonopoulos, C.P., Petropoulos, C, Antonopoulos, K Triantafyllou, V Voros, N.S, "The effect of symmetric block ciphers on WSN performance and behavior", International Conference on Wireless and Mobile Computing, Networking and Communications, October 2012.
10. Daojing He, Sammy Chan, Shaohua Tang, "A Novel and Lightweight System to Secure Wireless Medical Sensor Networks", Ieee journal of biomedical and health informatics, January 2014.
11. Xiaohua Luo, Kougen Zheng, Yunhe Pan, Zhaohui Wu, "Encryption algorithms comparisons for wireless networked sensors", IEEE international Conference on Systems, Man and Cybernetics, 2004.

12. A.Praveena, S.Devasena, K.M. Arivu Chelvan, "Achieving Energy Efficient and Secure Communication in Wireless Sensor Networks", In proceeding of: Wireless and Optical Communications Networks, 2006 IFIP International Conference on, Bangalore, 2006
13. Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, Niraj K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols", Mobile Computing, IEEE Transactions on, December 2005.
14. Daniel Tze Huei Lai, Rezaul Begg, Marimuthu Palaniswami, "Healthcare sensor networks-Challenges towards practical implementation", CRC Press Taylor &. Francis Group
15. Soham Banerjee, Divyashikha Sethia, Tanuj Mittal, Ujjwal Arora, Akash Chauhan, "Secure Sensor Node with Raspberry Pi", Impact 2013, 2013.
16. Mathew Greens, Attack of the week: RC4 is kind of broken in TLS, A Few thoughts on Cryptogaphic engineering: Attack of the week:RC4 is a kind of broken in TLS, (March 12, 2013), http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html (July 2, 2014)
17. RC4, RC4-Wikipedia-the free encyclopedia, http://en.wikipedia.org/wiki/RC4, (July 2, 2014)
18. Skipjack Review, Skipjack Review, http://www.austinlinks.com/Crypto/skipjack-review.html, (July 3, 2014)
19. Skipjack, Skipjack-everything2.com, http://everything2.com/title/Skipjack, (April 29, 2014)
20. The esp cbc-mode cipher algorithms, draft-ietf-ipsec-ciph-cbc-01: The esp cbc-mode cipher algorithms, (July 2, 1997) http://tools.ietf.org/html/draft-ietf-ipsec-ciph-rc5-cbc-00, (July 2, 2014)
21. Skipjack, Skipjack, http://www.cryptomuseum.com/crypto/usa/skipjack.htm (July 2, 2014)
22. Kenneth W Dam, Herbert S Lin, "Cryptography's Role in Securing the Information Society", By Committee to Study National Cryptography Policy, Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences, National Research Council,1996

23. US Congress, Office of Technology Assessment, "Issue Update on Information Security and Privacy in Network Environments", By DIANE Publishing Company, September 1995
24. Shish Ahmad, Mohd. Rizwan beg, Qamar Abbas, "Energy Efficient Sensor Network Security Using Stream Cipher Mode of Operation", Int'l Conf. on Computer & Communication Technology, 2010
25. Tingyuan Nie, Chuanwang Song, Xulong Zhi, "Performance Evaluation of DES and blowfish", Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on, 2010.
26. Introduction to cryptography, Introduction to cryptography, http://www.infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm (June 28, 2014)
27. Blowfish(cipher), Blowfish(cipher) Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Blowfish_(cipher) (july 2, 2014)
28. Coverity Scan: health-check, Coverity Scan-Static Analysis, (26 july 2013) https://scan.coverity.com/projects/661, (June 2014)
29. Coverity Scan: power-calibrate, Coverity Scan- Static Analysis, (26 july 2013), https://scan.coverity.com/projects/1732, (June 2014)
30. ColinKing, ColinKing-Ubuntu Wiki, (4 June 2014) https://wiki.ubuntu.com/ColinKing/, (June 2014)
31. Chou Fan, Jin Tan, Peng Zheng, "Low Speed Wireless networks research and simulation based on RC 5", Wireless communications, Networking and mobile computing, 2009, Wicom '09. 5th international conference on, September 2009.
32. A website www.stackoverflow.com,(July 2, 2014).
33. RC4 implementation from pseudocode, RC4 implementation from pseudocode, http://lists.runrev.com/pipermail/use-livecode/2007-July/101334.html(july 6, 2014)
34. The Rc4 stream encryption algorithm, http://cse.spsu.edu/afaruque/it6833/RC4.pdf, http://cse.spsu.edu/afaruque/it6833/RC4.pdf,
(July 8, 2014)
35. BLOWFISHENC: Blowfish Encryption Algorithm, BLOWFISHENC: Blowfish Encryption Algorithm(Theory)FPGA & Digital Design Lab, Computer science and engineering IIT Delhi Virtual Labs, http://iitd.vlab.co.in/?sub=66&brch=184&sim=1147&cnt=1, (July 8, 2014)

36. 0308feat2fig1.gif, 0308feat2fi,g1.gif(400*579) http://i.cmpnet.com/embedded/gifs/2003/0308/0308feat2fig1.gif,(July 8, 2014)
37. Konstantinos Papadopoulos,“ Implementation of security algorithms for wireless sensor networks using reconfigurable devices”, unpublished.
38. The Rc5 encryption algorithm, http://people.csail.mit.edu/rivest/Rivest-rc5.pdf, http://people.csail.mit.edu/rivest/Rivest-rc5.pdf, (June 28,2014)
39. Upload.wikimedia.org, http://upload.wikimedia.org/wikipedia/commons/7/7f/RC5_InfoBox_Diagram.svg(July 10, 2014)

APPENDIX

Here we have included some important pseudo-codes which we have Implemented/Used in this work.

1. **Pseudo Code for RC4 Algorithm:**

```
Function rc4 pText, pKey

  Initialize:

  repeat with i = 0 to 255
    put i into S1[i]
  end repeat

  put 0 into i
  repeat with n = 0 to 255
    add 1 to i
    if i > length(pkey) then put 1 into i
    put charToNum(char i of pKey) into S2[n]
  end repeat


  put 0 into j
  repeat with i = 0 to 255
    put (j + S1[i] + S2[i]) mod 256 into j
    put S1[i] into temp
    put S1[j] into S1[i]
    put temp into S1[j]
  end repeat
```

Encrypt/Decrypt:

```
  put 0 into i ; put 0 into j
  repeat for each char c in pText
    put charToNum(c) into tChar

    put (i + 1) mod 256 into i
    put (j + S1[i]) mod 256 into j
    put S1[i] into temp
    put S1[j] into S1[i]
    put temp into S1[j]
    put (S1[i] + S1[j]) mod 256 into t
    put S1[t] into K

    put numToChar(tChar bitXor K) after tOutput
  end repeat

  return binToHex(tOutput)
end rc4

function binToHex pString
```

```
  repeat for each char c in pString
    get charToNum(c)
    put baseConvert(it,10,16) into tTemp
    if it < 16 then put "0" before tTemp
    put tTemp after tHex
  end repeat
  return tHex
end binToHex
```

2. **Pseudo Code for RC5 Algorithm:**

```
//Key expansion
// Define two word-sized binary constants Pw and Qw
//Converting the Secret Key K[0...b-1] from Bytes to Words
c = [max(b, 1) / u]
for i = b - 1 downto 0 do
L[i / u] = (L[i / u] <<< 8) + K[i]
// Initialising S
S[0] = Pw;
for i = 1 to t - 1 do
S[ i ] = S[i - 1] + Qw;
// Mixing the secret key
i = j = 0;
a = b = 0;
```

```
do 3 * max(t, c) times:

a = S[i] = (S[i] + a + b) <<< 3;

b = L[i] = (L[j] + a + b) <<< (a + b);

i = (i + 1) mod (t);

j = (j + 1) mod (c);
```

Encryption:

```
A = A + S[0];

B = B + S[1];

for i = 1 to r do

A = ((A Xor B) <<< B) + S[ 2 * i ]

B = ((B Xor A) <<< A) + S[ 2 * i + 1]
```

Decryption:

```
for i = r down to 1 do

B = ((B - S[2 * i + 1] >>> A) Xor A;

A = ((A - S[2 * i] >>> B) Xor B;

B = B - S[1];

A = A - S[0];
```

3. **Pseudo Code for Skipjack Algorithm:**

Function Skipjack

//G on a sub-word (4 bytes) is a 4 round feistal structure

Encryption:

Input w_i^o, 1<=i<=4;

```
Counter=0;

Do counter =1 to 8;

    Rule A
    $w_1^{k+1} = G^k(w_1^k) \oplus w_4^k \oplus counter^k$
    $w_2^{k+1} = G^k(w_1^k)$
    $w_3^{k+1} = w_2^k$
    $w_4^{k+1} = w_3^k$

Counter++;

Do counter=8 t0 16;

    Rule B
    $w_1^{k+1} = w_4^k$
    $w_2^{k+1} = G^k(w_1^k)$
    $w_3^{k+1} = w_1^k \oplus w_2^k \oplus counter^k$
    $w_4^{k+1} = w_3^k$

Counter++;

Do counter = 16 to 24;

Repeat Rule A;

Counter ++;

Do counter= 24 to 32;

Repeat Rule B;

Counter++;

Output=$w_i^{32}$,1<=i<=4;

Decryption

Input= $w_i^{32}$,1<=i<=4;
```

```
Counter=32;

Do counter= 32 to 24;

    Rule $B^{-1}$
    $w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$
    $w_2^{k-1} = [G^{k-1}]^{-1}(w_2^k) \oplus w_3^k \oplus counter^{k-1}$
    $w_3^{k-1} = w_4^k$
    $w_4^{k-1} = w_1^k$

Counter--;

Do counter= 24 to 16;

    Rule $A^{-1}$
    $w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$
    $w_2^{k-1} = w_3^k$
    $w_3^{k-1} = w_4^k$
    $w_4^{k-1} = w_1^k \oplus w_2^k \oplus counter^{k-1}$

Counter--;

Do counter=16 to 8;

Repeat rule $B^{-1}$;

Counter--;

Do counter= 8 to 0;

Repeat rule $A^{-1}$;

Counter--;

Output= $w_i^o$, 1<=i<=4;
```

4. **Pseudo Code for Blowfish Algorithm:**

```
//Key expansion-converts a key of at most 448 bits into several subkey arrays (total 4168 bytes)

Define P-array 18, 32-bit subkeys:P1,P2,………….,P18

Define Four 32-bit S-Boxes,256 entries each:

S1,0, S1,1,………. S1,255

S2,0, S2,1,……….. S2,255

S3,0, S3,1,……….. S3,255

S4,0, S4,1,.............S4,255

//Generating the Subkeys:

 Initialize P-array, initialise four S-boxes using string(value->hexadecimal digits of pi(less the initial 3))→ 1

P1 = 0x243f6a88, P2 = 0x85a308d3, P3 = 0x13198a2e, P4 = 0x03707344, etc. →1

 for all bits of key(repeat until all p array XORed with key)→ 2

P1 XOR (first) 32 bits of key→2

P2 XOR (second)32-bits of the key→2

 Encrypt the all-zero string with the Blowfish algorithm,(using the subkeys generated in steps (1) and (2))→3

 P1,P2=output of step (3).

 Encrypt output of step (3) using the Blowfish algorithm(with the modified subkeys)→ 5

 P3,P4=output of step (5).

 repeat replacing all entries of the P array

Repeat for all four S-boxes
```

Encryption:

```
Rounds: 16

Input: 64 bit data element->x
```

Divide x into two 32-bit halves: xL, xR.

Then, for i = 1 to 16:

xL = xL XOR Pi

xR = F(xL) XOR xR

Swap xL and xR

After the sixteenth round, swap xL and xR again to undo the last swap.

Then, xR = xR XOR P17 and xL = xL XOR P18. Finally, recombine xL & xR to get the cipher text