4.1. Introduction

Near Field Communication (NFC) is a short-range wireless connectivity technology that enables exchange of data between devices and tags over a distance of up to about 10 centimeters. It’s mainly aimed at making it easier to use services for payment, public transport and data sharing between devices but a number of other uses have been proposed [WIK01, NFC01]:

- Identity documents
- Health monitoring & Identification of medication
- Mobile commerce, guiding of consumers in retail
- Time and attendance applications
- Electronic keys - car keys, house/office keys, and hotel room keys, etc.

Configuration and initiation of other wireless network connections such as Bluetooth, Wi-Fi or Ultra-wideband
4.1.1 Features of NFC

NFC can be used in both read and write mode, operating at 13.56 MHz with data transfer rates at 106, 212 or 424 Kbits/second. Communication between two NFC-compatible devices occurs when they are brought within about 5 centimeters of one another. A swipe or touch can establish an NFC connection. The underlying layers of NFC technology follow universally implemented ISO, ECMA, and ETSI standards.

4.1.2 The NFC Tag

The NFC tag is a passive component that usually store data that can be read by an NFC-enabled device via a temporary wireless connection. The NFC Forum have identified four tag types with different properties regarding data transfer rate, storage capability, security and price [NFC02]. The tags are manufactured in different size and shape depending on the application; in Figure 4.1 some examples are shown.

Figure 4.1: NFC tags
4.1.3 NFC Devices

The NFC technology may be implemented in a number of everyday devices to make them be able to communicate with NFC tags or each other, e.g. phones, digital cameras, vending machines, ATMs or parking meters. NFC enabled devices may change operating mode between the following:

- Reader/Writer - the NFC Device is able to read and write to tags.
- Peer-to-Peer - allows two devices to exchange data.
- Card Emulation - the device itself can act as an NFC tag.

This thesis will only deal with the use of the Read/Write mode on tags and the interaction with the passive tags on card emulation.
4.2 ACR122 CARD READER

The ACR122 is a PC-linked Contactless Smart Card Reader/Writer developed on the 13.56MHz Contactless Technology. It is the world’s first CCID Compliant Contactless Card Reader/Writer that follows both ISO14443 and ISO18092. This device is designed to support not only MiFare and ISO14443 Type A and B Cards but also FeliCa and NFC tags.

By making use of up to 424 Kbps for NFC Tags access and full USB speed of up to 12 Mbps, ACR122 can read and write faster and more efficiently. Furthermore, ACR122 is also PC/SC compliant which allows interoperability for
different applications development. With its compact size and trendy design and with the various features the ACR122 offers, we can experience the convenience in using ACR122 for applications of payment, mass transit, access control, time attendance, etc, as illustrated in Figure 4.2.

4.2.1 Features

- CCID Standard
- PC/SC compliant
- Read / Write Speed up to 424kbps
- ISO/IEC18092 (NFC) compliant
- ISO14443 Type A and B card support
- Support Mifare® card
- Support FeliCa card
- USB PnP
- Bi-Color LED
- Compact size: 98mm (L) x 65mm (W) x 12.8mm (H)
- Light Weight: 70g
- CE, FCC and RoHS compliant
- User Controllable Buzzer (optional)
4.2.2 Typical Applications

- Network access control
- Micro-payment
- NFC mobile tag
- Public Transportation Terminals
- Automatic Fare Collection
- Physical access control
- Customer Loyalty
- Time attendance
- Contactless public phones
- Vending machines

4.3 MIFARE - Introduction

An RFID card gets more and more common. In general, two types of RFID cards are used, active and passive cards. While active systems have an own source of energy, passive systems rely on the energy provided by the card readers.

One of the most common RFID Cards is the MIFARE Classic, produced by NXP Semiconductors.

The MIFARE offers different types of cards, which are all passive and fully compliant with ISO/IEC 14443 Type A.
4.3.1 MIFARE

There are two common types of the MIFARE Classic cards, the MIFARE Classic 1k and 4k. The 1k Chip has 1k EEPROM memory, which is separated in 16 sectors with 4 blocks, each containing 16 byte. This makes a total of 64 blocks. The 4k version offers 4k of EEPROM memory, separated in 256 blocks, where 32 sectors have 4 blocks and additional 8 sectors having 16 blocks.

This thesis will only deal with MIFARE Classic 1k card.

4.3.2 MIFARE Classic 1K

Features and benefits

- Contactless transmission of data and supply energy
- Operating distance up to 100 mm depending on antenna geometry and reader configuration
- Operating frequency of 13.56 MHz
- Data transfer of 106 Kbit/s
- Data integrity of 16-bit CRC, parity, bit coding, bit counting
- Anti-collision

EEPROM

- 1 KB, organized in 16 sectors of 4 blocks (one block consists of 16 byte)
- User definable access conditions for each memory block
- Data retention time of 10 years
- Write endurance 100,000 cycles
4.3.2.1 Block Diagram

Block diagram of Mifare Card, as illustrated in figure 4.3 below.
4.3.3 MiFare 1K card – Memory organization

The 1024 x 8 bit EEPROM memory is organized in 16 sectors with 4 blocks of 16 bytes each.

In the erased state the EEPROM cells are read as a logical '0', in the written state as a logical '1'.

![Figure 4.4 Mifare Card – Memory Organization](image)

MIFARE 1K Memory Map.

<table>
<thead>
<tr>
<th>Sectors</th>
<th>Data Blocks</th>
<th>Trailer Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sector 0</td>
<td>0x00 ~ 0x02</td>
<td>0x03</td>
</tr>
<tr>
<td>Sector 1</td>
<td>0x04 ~ 0x06</td>
<td>0x07</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sector 14</td>
<td>0x36 ~ 0x0A</td>
<td>0x35</td>
</tr>
<tr>
<td>Sector 15</td>
<td>0x3C ~ 0x3E</td>
<td>0x3F</td>
</tr>
</tbody>
</table>

![Figure 4.5 Mifare Card – Memory Map](image)
4.4 BASIC ARCHITECTURE COMPONENTS

A basic architecture framework defines how to create and use an Electronic Medical Records in hospital/clinical is illustrated in Figure 4.5.
4.4.1 COMPONENTS - ASSUMPTIONS

Hospital/clinic - with EMR system: standardized supporting HL7 protocol

NFC tags - with EMR details for every patient

NFC Enabled devices to read/update EMR details

4.4.2. TYPICAL OPERATION FLOW

**Hospital/Clinical EMR:** Its holds patient EMR

**Visits/Admit:** receptionist/admin/doctor synchronizes the patient EMR with his EMR system by a swipe or bump with patient NFC Tag/Device

**Access EMR:** physician/doctor/providers view latest patient EMR

**Lab Reports:** Lab orders/Reports updated with NFC tags instead of barcodes/papers

**Inpatient routine updates:** Every day inpatient chart updates/lab orders/medications are done with NFC tags / NFC enabled devices - like Smartphone’s/tablets

**Discharge/Leave:** Synchronize patient NFC tag - updates current visit information into patient EMR
4.4.3 ARCHITECTURE NFC COMPONENTS

Figure 4.6. Architecture NFC Components
4.4.4 NFC COMPONENT

**Pack/Unpack:** Responsible for encoding/decoding the patient EMR as compressed raw data.

*Classes:* Blowfish, Huffman - utility class

**Data:** Actual patient EMR used in patient care

*Classes:* PatientEMR, BaseModel

**Authentication:** Responsible for security - allows only authorized users/systems to update NFC devices/tags

*Classes:* Mifare1KCard

**Updater:** updates patient EMR in current NFC tag

*Classes:* Mifare1KCard, INFCCard, JacspcsLoader

**Generator:** Creates patient EMR in current NFC tag

*Classes:* Mifare1KCard, INFCCard, JacspcsLoader

**Electronic Medical Records EMR:** sample interface for view/updating patient EMR details

*Classes:* EMRInterface
4.5 ALGORITHM

4.5.1 Algorithm - Read PatientEMR:

Step 1: Initialize NFC reader

Step 2: Read Raw data from NFC card/tag with ReadCard Algorithm

Step 3: Convert raw data to PatientEMR with Data Decoder Algorithm

Step 4: Populate the EMRPanel/UI with the PatientEMR model created

- Populate the base EMR details (Refer the EMRUtility for converting ICD/SNOMED Codes with description)
- Loop through the List of Visits
  - Populate the visit details
  - Populate the Vitals details of the Visit

This algorithm starts by initializing the NFC reader device. The raw encrypted/encoded data is read from the device with the ReadCard Algorithm defined below. The raw data is later converted into PatientEMR model with the Data Decoder Algorithm defined below. Patient EMR information is sent to/updated in the host system - in this case from the EMR UI and a new PatientEMR model is created. While doing so it uses ICD/SNOMED codes for representing the actual diagnosis and medications in the EMR through utility functions. Then entire list of visits are looped from the PatientEMR model and populated in UI along with the Vitals details.
4.5.2 Algorithm - WritePatientEMR:

Step 1: Initialize NFC reader

Step 2: Create a PatientEMR model and populate details from EMRPanel/UI
   - Populate the base EMR details (Refer the EMRUtility for converting ICD/SNOMED Codes with description)
   - Loop through the List of Visits
     o Populate the visit details
     o Populate the Vitals details of the Visit

Step 3: Convert PatientEMR to raw data to with Data Encoder Algorithm

Step 4: Write raw data to NFC card/tag with WriteCard Algorithm

This algorithm starts by initializing the NFC reader device. Then the patient EMR information is fetched from the host system - in this case from the EMR UI and a new PatientEMR model is created. While doing so it uses ICD/SNOMED codes for representing the actual diagnosis and medications in the EMR through utility functions. Then entire list of visits are looped from the UI and populated in the PatientEMR model along with the Vitals details.

This PatientEMR model is converted into raw data with encoder and encryption algorithm defined in the following sections. Later the raw data is written on the NFC card/device transparently to the user with our WriteCard algorithm defined in following sections.
4.5.3 Algorithm - ReadCard

Step 1: Initialize the card reader

- Establish context and obtain handle for further manipulation
- List PC/SC card readers installed in the system
- Connect the ACR card reader

Step 2: Initialize the 1K buffer – for reading EMR details

Step 3: For all Block 1 to 64 (4 Sectors - 16 Blocks each) - MiFare 1K card has 64 blocks

Step 4: Skip the 0 - 3 Blocks, as they have Manufacturer details and other card information

Step 5: For other Blocks (data blocks) of the NFC card

- Authenticate - block
- Clear Input / Output buffer to read/write from card reader
- Read block data (16 bytes information) with “Read Binary Blocks command”
- On successful read operation - copy data into buffer

This algorithm starts by initializing the NFC Card reader attached with the hospital/clinic systems. Initialization is done by establishing the context and obtaining the handle for further manipulation. It lists the PC/SC card readers installed in the system. Later the default card reader is connected. Later 1 K buffer is initialized for further reading from the device – this is available in user space for application to read/write data in and out of device.

As defined in the memory architecture of MiFare card has 64 blocks grouped as 4 sectors with 16 blocks each. The first 4 blocks of sector 1 contains
manufacturer details of the NFC card/tag and hence it is skipped for read/write operations. Rest of the blocks contains the actual raw data.

Every block read are authenticated and then user buffers (both input and output) are initialized with default data or data to be communicated. MiFare and related devices uses flash technology to store and retrieve information and allow block by block transfers. Given the base command to read 16 byte block information and block number – device transfer raw data into buffer allocated on success. Later all the block information is combined to form the raw PatientEMR information.

4.5.4 Algorithm - WriteCard

Step 1: Initialize the card reader

- Establish context and obtain handle for further manipulation
- List PC/SC card readers installed in the system
- Connect the ACR card reader

Step 2: Initialize the 1K buffer – for reading EMR details

Step 3: For all Block 1 to 64 (4 Sectors - 16 Blocks each) - MiFare 1K card has 64 blocks

Step 4: Skip the 0 - 3 Blocks, as they have Manufacturer details and other card information

Step 5: For other Blocks (data blocks) of the NFC card

- Authenticate - block
- Clear Input/Output buffer to read/write from card reader
- Write block data (16 bytes information) with “Update Binary Blocks command”

This algorithm starts by initializing the NFC Card reader attached with the hospital/clinic systems. Initialization is done by establishing the context and obtaining the handle for further manipulation. It lists the PC/SC card readers installed in the system. Later the default card reader is connected. Later 1 K buffer is initialized for further reading from the device – this is available in user space for application to read/write data in and out of device.

As defined in the memory architecture of MiFare card has 64 blocks grouped as 4 sectors with 16 blocks each. The first 4 blocks of sector 1 contains manufacturer details of the NFC card/tag and hence it is skipped for read/write operations. Rest of the blocks contains the actual raw data.

Every block read are authenticated and then user buffers (both input and output) are initialized with default data or data to be communicated. Raw PatientEMR is broken into 16 byte chunks and with the base command to write 16 byte block information and block number – device writes raw data from buffer into the NFC card/tag. On successful completion the device sets success flag, which is read from the application and user is informed accordingly.
4.5.5 Algorithm – Data Decoder

Step 1: With the raw data from NFC enabled device/tag - Apply decryption (2-way blowfish algorithm) and decoding (Huffman algorithm) to convert into stream of data

Step 2: With Reflection create an Instance of the PatientEMR model

Step 3: Fetch the fields list from PatientEMR model

Step 4: For each field in the fields list

- Read stream data and uncompress it
- Populate model with the uncompressed data

Step 4: Return the object

This algorithm takes raw data read from NFC card/tag as input and converts it to PatientEMR model object. On the input raw data a 2-way blowfish algorithm (modified blowfish) is applied for decryption and then the result data is decoded with standard Huffman algorithm. This decoded info is nothing but the raw serialized PatientEMR.

We later apply Java’s Reflection APIs to create PatientEMR model. Initially the model object is created and by looping through all the fields in the class and by reading the corresponding stream data we get actual data of the field. We populate entire PatientEMR model after extracting data in similar fashion. On successful completion of iteration - the model object is returned to user.
4.5.6 Algorithm – Data Encoder

Step 1: With Reflection Fetch the fields list from PatientEMR model

Step 2: For each field in the fields list
- Read the field information from the model
- Compress the data and pack the stream of information

Step 3: Call security module with the stream of information obtained

Step 4: With the stream data - Apply encryption (2-way blowfish algorithm) and encoding (Huffman algorithm) to obtain the data to be stored in NFC enabled device/tag

This algorithm takes PatientEMR model object as input and converts it to raw data to be written to NFC card/tag. With the help of Java’s Reflection APIs PatientEMR object details are extracted and a compressed serialized version of the object is created iteratively repeating the same 2 steps. i.e by reading the field information and writing the compressed data in the buffer.

Later the serialized data is sent to security module for encryption and encoding. Security module in turn does the following actions. On the input raw data 2-way blowfish algorithm (modified blowfish) is applied for encryption and then the result data is encoded with standard Huffman algorithm. On successful completion of encoding – raw data is returned to user.
4.6 Comparison For different algorithm and used in our implementation

4.6.1 Standard Algorithm Used

As the importance and the value of exchanged data over the internet or other media types are increasing, the search for the best solution to offer the necessary protection against the data thieves’ attacks along providing these services under timely manner is one of the most active subjects in the security related communities.

The comparison between the most common and used algorithms in the data encryption field. Since our main concern here is the performance of these algorithms comparison takes into consideration the behavior and the performance of the algorithm when different data loads are used.

4.6.2 Cryptography: Overview

An overview of the main goals behind using cryptography will be discussed in this section along with the common terms used in this field.

Cryptography is usually referred to as "the study of secret", while nowadays is most attached to the definition of encryption. Encryption is the process of converting plain text "unhidden" to a cryptic text "hidden" to secure it against data thieves. This process has another part where cryptic text needs to be
decrypted on the other end to be understood. Figure 4.7. shows the simple flow of commonly used encryption algorithms.

![Encryption-Decryption Flow](image)

**Figure 4.7. Encryption-Decryption Flow**

Cryptographic system is defined as "a set of cryptographic algorithms together with the key management processes that support use of the algorithms in some application context." This definition defines the whole mechanism that provides the necessary level of security comprised of network protocols and data encryption algorithms.

**4.6.2.1 Cryptography Goals**

Every security system must provide a bundle of security functions that can assure the secrecy of the system. These functions are usually referred to as the goals of the security system. These goals can be listed under the following five main categories:

**Authentication:** This means that before sending and receiving data using the system, the receiver and sender identity should be verified.
Secrecy or Confidentiality: Usually this function (feature) is how most people identify a secure system. It means that only the authenticated people are able to interpret the message (date) content and no one else.

Integrity: Integrity means that the content of the communicated data is assured to be free from any type of modification between the end points (sender and receiver). The basic form of integrity is packet check sum in IPv4 packets.

Non-Repudiation: This function implies that neither the sender nor the receiver can falsely deny that they have sent a certain message.

Service Reliability and Availability: Secure systems usually get attacked by intruders, which may affect their availability and type of service to their users. Such systems should provide a way to grant their users the quality of service they expect.

4.6.3 Standard Cryptography Compared Algorithms

DES: (Data Encryption Standard), was the first encryption standard to be recommended by NIST (National Institute of Standards and Technology). It is based on the IBM proposed algorithm called Lucifer. DES became a standard in 1974. Since that time, many attacks and methods were recorded that exploit the weaknesses of DES, which made it an insecure block cipher.
3DES: As an enhancement of DES, the 3DES (Triple DES) encryption standard was proposed. In this standard the encryption method is similar to the one in original DES but applied 3 times to increase the encryption level. But it is a known fact that 3DES is slower than other block cipher methods.

AES: (Advanced Encryption Standard), is the new encryption standard recommended by NIST to replace DES. Rijndael (pronounced Rain Doll) algorithm was selected in 1997 after a competition to select the best encryption standard. Brute force attack is the only effective attack known against it, in which the attacker tries to test all the characters combinations to unlock the encryption. Both AES and DES are block ciphers.

Blowfish: It is one of the most common public domain encryption algorithms provided by Bruce Schneier - one of the world's leading cryptologists, and the president of Counterpane Systems, a consulting firm specializing in cryptography and computer security.

Blowfish is a variable length key, 64-bit block cipher. The Blowfish algorithm was first introduced in 1993. This algorithm can be optimized in hardware applications though it is mostly used in software applications. Though it suffers from weak keys problem, no attack is known to be successful against it.

Brief descriptions of the compared encryption algorithm have been introduced. These introductions to each algorithm provide the minimum information to distinguish the main differences between them. Table 4.1 below
contains the speed benchmarks for some of the most commonly used cryptographic algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Megabytes($2^{20}$ bytes) Processed</th>
<th>Time Taken</th>
<th>MB/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish</td>
<td>256</td>
<td>3.976</td>
<td>64.386</td>
</tr>
<tr>
<td>Rijndael (128-bit key)</td>
<td>256</td>
<td>4.196</td>
<td>61.01</td>
</tr>
<tr>
<td>Rijndael (192-bit key)</td>
<td>256</td>
<td>4.817</td>
<td>53.145</td>
</tr>
<tr>
<td>Rijndael (256-bit key)</td>
<td>256</td>
<td>5.308</td>
<td>48.229</td>
</tr>
<tr>
<td>Rijndael (128) CTR</td>
<td>256</td>
<td>4.436</td>
<td>57.71</td>
</tr>
<tr>
<td>Rijndael (128) OFB</td>
<td>256</td>
<td>4.837</td>
<td>52.925</td>
</tr>
<tr>
<td>Rijndael (128) CFB</td>
<td>256</td>
<td>5.378</td>
<td>47.601</td>
</tr>
<tr>
<td>Rijndael (128) CBC</td>
<td>256</td>
<td>4.617</td>
<td>55.447</td>
</tr>
<tr>
<td>DES</td>
<td>128</td>
<td>5.998</td>
<td>21.34</td>
</tr>
<tr>
<td>(3DES)DES-XEX3</td>
<td>128</td>
<td>6.159</td>
<td>20.783</td>
</tr>
<tr>
<td>(3DES)DES-EDE3</td>
<td>64</td>
<td>6.499</td>
<td>9.848</td>
</tr>
</tbody>
</table>

The popular secret key algorithms including DES, 3DES, AES (Rijndael), Blowfish, were implemented, and their performance was compared by encrypting input files of varying contents and sizes. The algorithms were implemented in a uniform language (Java), using their standard specifications, and were tested on two different hardware platforms, to compare their performance.
Table 4.2 Execution Time Comparisons

<table>
<thead>
<tr>
<th>Input Size (bytes)</th>
<th>DES</th>
<th>3DES</th>
<th>AES</th>
<th>BF</th>
<th>2-BF</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,527</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>36,002</td>
<td>4</td>
<td>13</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>45,911</td>
<td>5</td>
<td>17</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>59,852</td>
<td>7</td>
<td>23</td>
<td>11</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>69,545</td>
<td>9</td>
<td>26</td>
<td>13</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>137,325</td>
<td>17</td>
<td>51</td>
<td>26</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>158,959</td>
<td>20</td>
<td>60</td>
<td>30</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>166,364</td>
<td>21</td>
<td>62</td>
<td>31</td>
<td>17</td>
<td>34</td>
</tr>
<tr>
<td>191,383</td>
<td>24</td>
<td>72</td>
<td>36</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>232,398</td>
<td>30</td>
<td>87</td>
<td>44</td>
<td>24</td>
<td>48</td>
</tr>
<tr>
<td>Average Time</td>
<td>14</td>
<td>42</td>
<td>21</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>Bytes/sec</td>
<td>7,988</td>
<td>2,663</td>
<td>5,320</td>
<td>10,167</td>
<td>20334</td>
</tr>
</tbody>
</table>

From above Table 4.2 Comparative execution times (in seconds) of encryption algorithms in P-4 2.4 GHz machine.

From the results it is easy to observe that Blowfish has an advantage over other algorithms in terms of throughput.

The results showed that Blowfish has a very good performance compared to other algorithms. Also it showed that AES has a better performance than 3DES and DES. Amazingly it shows also that 3DES has almost 1/3 throughput of DES, or in other words it needs 3 times than DES to process the same amount of data.
Experiments for comparing the performance of the different encryption algorithms implemented inside J2EE framework. Their results are close to the ones shown in Figure 4.5.

The comparison was performed on the following algorithms: DES, Triple DES (3DES), RC2 and AES (Rijndael). The results show that AES outperformed other algorithms in both the number of request processes per second in different user loads, and in the response time in different user-load situations.
Our Implementation is 2-way Blowfish Algorithm.

Modified standard usage of Blowfish Algorithm to provide additional security.

- First pass with an application level.
- Second pass with user private key.

4.6.4 Data Compression – Encoding / Decoding

Why we need data compression – lesser disk space (more data in reality), faster writing and reading, faster file transfer, variable dynamic range and byte order independent.

Compression Algorithm Compared

The following data compression algorithms for compression ratio and time compression on a variety of raw detector images from different scientific field were compared. As illustrated in Table 4.3. And Table 4.4
Table 4.3. Compression times

<table>
<thead>
<tr>
<th>Mbytes</th>
<th>2.73</th>
<th>2.75</th>
<th>7.63</th>
<th>8.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.O.</td>
<td>0.73</td>
<td>0.77</td>
<td>2.10</td>
<td>3.45</td>
</tr>
<tr>
<td>B.O/pack</td>
<td>1.86</td>
<td>1.72</td>
<td>4.32</td>
<td>4.45</td>
</tr>
<tr>
<td>B.O/gzip</td>
<td>4.14</td>
<td>4.01</td>
<td>9.97</td>
<td>14.7</td>
</tr>
<tr>
<td>Pack</td>
<td>1.67</td>
<td>1.67</td>
<td>4.11</td>
<td>3.95</td>
</tr>
<tr>
<td>Compress</td>
<td>3.60</td>
<td>3.75</td>
<td>8.89</td>
<td>8.01</td>
</tr>
<tr>
<td>Gzip</td>
<td>8.85</td>
<td>6.23</td>
<td>16.9</td>
<td>33.0</td>
</tr>
<tr>
<td>Bwt</td>
<td>163</td>
<td>177</td>
<td>436</td>
<td>406</td>
</tr>
<tr>
<td>ha 1</td>
<td>25.8</td>
<td>34.9</td>
<td>86.5</td>
<td>131</td>
</tr>
<tr>
<td>ha 2</td>
<td>54.8</td>
<td>106.9</td>
<td>262</td>
<td>89.1</td>
</tr>
</tbody>
</table>

Table 4.4. Compression Ratios (Input/Output Sizes)

<table>
<thead>
<tr>
<th>Mbytes</th>
<th>2.73</th>
<th>2.75</th>
<th>7.63</th>
<th>8.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.O.</td>
<td>1.99</td>
<td>1.84</td>
<td>1.93</td>
<td>1.98</td>
</tr>
<tr>
<td>B.O/pack</td>
<td>2.62</td>
<td>2.2</td>
<td>2.43</td>
<td>3.54</td>
</tr>
<tr>
<td>B.O/gzip</td>
<td>2.59</td>
<td>2.41</td>
<td>2.71</td>
<td>3.32</td>
</tr>
<tr>
<td>Pack</td>
<td>1.61</td>
<td>1.42</td>
<td>1.56</td>
<td>2.01</td>
</tr>
<tr>
<td>Compress</td>
<td>1.72</td>
<td>1.69</td>
<td>1.81</td>
<td>2.69</td>
</tr>
<tr>
<td>Gzip</td>
<td>1.77</td>
<td>1.71</td>
<td>1.85</td>
<td>2.56</td>
</tr>
<tr>
<td>Bwt</td>
<td>2.17</td>
<td>1.91</td>
<td>2.02</td>
<td>2.66</td>
</tr>
<tr>
<td>ha 1</td>
<td>1.99</td>
<td>1.84</td>
<td>1.93</td>
<td>3.82</td>
</tr>
<tr>
<td>ha 2</td>
<td>2.62</td>
<td>2.2</td>
<td>2.43</td>
<td>3.01</td>
</tr>
</tbody>
</table>
Our Implementation: pack – standard Huffman Implementation

With the above comparison work we choose standard Huffman encoding

- **No Patent Problems:** Dictionary compression and arithmetic compression are both areas of legal problems over patents.

- **Fast:** Static Huffman compression is much faster for both coding and decoding than arithmetic compression. (See "Lossless Compression for Text and Images", A Moffat, T C Bell, and I H Witten, in "Signal Compression: Coding of Speech, Audio, Text, Image, and Video", published by World Scientific Publishing, page 195-196, 1997.)

- **Practical:** The compression algorithm comparisons show that Huffman based algorithms give good compression at a reasonable time.

- **Very Close to Optimum:** Whilst Huffman encoding must assign an integer number of bits to every symbol, this is not a limitation to compression ratios in almost all practical cases.

- **Static Algorithm:** The coding table can be produced once and used for many images with the same statistical characteristics, to avoid the need of calculating image statistics and symbol codes.

- **Entry Look-Up Table:** Since very pixel is coded by 1 of more bits, if predictor 1 is used, a simple look-up table may be defined between pixel position in the array and bit position in the compressed data.
4.6.5 Flash Memory (Complete) Read Write Algorithms.

Flash memory is a type of electrically erasable programmable read-only memory (EEPROM), because flash memories are non volatile and relatively dense, they are now used to store files and other objects in handheld computers, mobile phones, digital cameras, portable music players and many other computer systems in which magnetic disk are inappropriate.

Flash, like earlier eeprom devices, suffers from two limitations.

- First, bits can only be cleared by erasing a large block of memory.
- Second, each block can only sustain a limited number of erasures, after which it can no longer reliably store data.

Our Implementation – Sequential Block Read/Write

Mifare 1k card has 64 blocks – with standard sequential block reading with authentication other flash memory read/write algorithm are analyzed but not used as EMR data and accessed from NFC tags/devices every time. We are proposing to use NFC tags/devices for synchronizing with hospital/clinical EMR at the time of admission and discharge.

Security Aspects:

- Access to data block – every read/write operation needs authentication
- Operating System/file system – data in device is encoded and stored as raw data.

Our implementation: sample coding and screenshots available in appendices.