UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATION ENGINEERING

Mohammed Assim Mohammed

PERFORMANCE ANALYSIS OF SECURITY MEASURES IN NEAR FIELD COMMUNICATION

Master´s thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, xx May, 2014.

Supervisor Prof. Timo Mantere

Instructor Tobias Glocker

ACKNOWLEDGEMENT

My sincere gratitude goes to Mr. Tobias Glocker, my instructor who has been a great help to have in this project appear the way it is. Without him it would not be possible to have this work done. Also I'd like to thank all the staff at the University of Vaasa for providing me with the valuable knowledge that hugely participated in the creation of this work. Finally thank all the people who have supported me in more ways than I could imagine. My family for their never ending love and support, my friends for believing in, and encouraging me to reach my goals. I am forever grateful.

Table of Contents

Table of Figures

Table of Tables

List of Abbreviation

3DES	Triple Data Encryption Standard
AC	Alternating Current
AES	Advanced Encryption Standard
ASK	Amplitude Shift Keying
Cm	Centimeter
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CICC	Close Inductive Coupling Card
CISC	Complex Instruction Set Computer
CSMA	Carrier Sense Multiple Access
DEP	Data Exchange Protocol
DES	Data Encryption Standard
DMA	Direct Memory Access
ECC	Elliptic Curves Cryptography
EPC	Electronic Product Code
ETSI	European Telecommunication Standards Institute
ECMA	European Computer Manufacturers Association
EEPROM	Electrically Erasable Read Only Memory
FCC	Federal Communication Commission
FRAM	Ferroelectric Random-Access Memory
GF	Galois Field
GHz	Giga Hertz
HF	High Frequency
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IDEA	International Data Encryption Algorithm
I/O	Input/Output
KHz	Kilo Hertz
Kbps	Kilo Bit Per Second
LF	Low Frequencies
LEN	Length
Man	Manchester Coding
MHz	Mega Hertz
Mod	Modified
MSB	Most Significant Byte
MMC	MultiMedia Card
MMU	Memory Management Unit
MicroSD	Micro Secure Digital
NFC	Near Field Communication

NPU	Numerical Processing Unit
NFCID	Near Field Communication Identifier
NFCIP-1	Near Field Communication Interface and Protocol-1
NFCIP-2	Near Field Communication Interface and Protocol-2
OS	Operating System
PCD	Proximity Coupling Device
PICC	Proximity Inductive Coupling Card
RF	Radio Frequency
RAM	Random Access Memory
RFU	Reserved for Future Use
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RNG	Random Number Generator
RFCA	Radio Frequency Collision Avoidance
RFID	Radio Frequency Identification
SE	Secure Element
SDD	Single Device Detection
SIM	Subscriber Identity Module
SWP	Single Wire Protocol
SYNC	Synchronization
TDMA	Time Division Multiple Access
UHF	Ultra High Frequency
USB	Universal Series Bus
USIM	Universal Subscriber Identity Module
VICC	Vicinity Inductive Coupling Card
WLAN	Wireless Local Area Network

University Of Vaasa

Faculty of Technology

Author:	Mohammed Assim Mohammed
Topic of the Thesis	Performance Analysis of Security Measure in Near Field Communication
Name of the Supervisor	Timo Mantere
Instructor:	Tobias Glocker
Degree:	Master of Science
Department:	Department of Computing Science
Degree Program:	Degree Program in Information Technology
Major Subject:	Telecommunication Engineering
Year of Entering the University:	2011
Year of Completing the Thesis:	2014 Page: 116

Abstract

Nowadays near field communication are largely used in so many different applications for the convenience and ease of use they provide. They store and exchange many personal data, some of them requires more security than others, due to the value they poses, such as banking information and personal identification. And maintaining high level of security is task of the utmost priority.

The main focus of this thesis is establishing a knowledge base for different NFC/RFID devices. Evaluating the different encryption algorithms used currently, based on their encryption/decryption time, their immunity to brute force attack, and the amount of power needed to execute them.

The encryption algorithms will be implemented using Python programing language and tested on a windows computer in order to test their immunity against brute force attack. Encryption/decryption time and the power usage will be tested on a Raspberry Pi, for the similarities it has with modern mobile devices.

Keywords: Encryption, Decryption, Brute Force Attack, NFC, Security.

1. Introduction

Ever since Marconi proved the radio waves capability to provide a continuous contact in 1897, the wireless telecommunication revolution started (Hioki 2000). The speed at which it grows has never been faster. Nowadays telecommunication has different applications such as broadcasting to several receivers (TV and Radio), point to point systems (control unit to machine), a point to multi point systems (cellular systems), wireless networks and many others varying from long to short distance. The main focus of this thesis is on Near Field Communication (NFC), a form of a short range communication, analyzing the security techniques used.

NFC technology was developed by Sony and Philips - based on radio frequency identification technology (RFID) - in order to share information at a maximum data rate of 424kbps by providing a wireless communication link between devices separated by less than 4 centimeters. NFC technology is compliant with RFID standards and protocols.

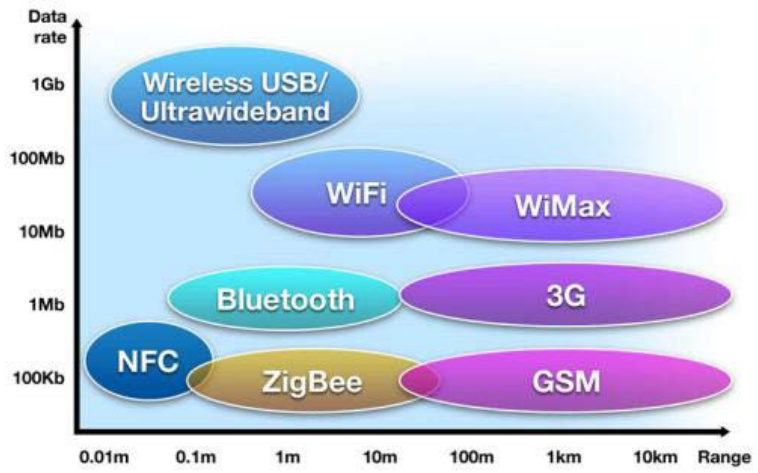

Figure 1. Distance and data rate difference of NFC with other existing wireless technologies (NFC Forum)

The communication between the two devices requires at least one of them to be active. Due to the simplicity of establishing a connection, this technology has potentials in various applications such as ticketing, e-commerce, electronics keys and identifications, and also complimenting many other wireless technologies such as Bluetooth and Wi-Fi.

NFC is compatible with other existing contactless infrastructure and it enables users to use one device with different systems (NFC Specifications). NFC is based on RFID technology, the main characteristic that differentiates NFC from RFID is that the new technology prepares bidirectional data transmission between NFC equipped devices. For the communication between the two devices it is just enough to bring them close together or make them touch physically (Hossein 2012).

The main objective of this thesis is to analyze and examine the different security techniques used, to ensure the integrity and safety of the shared information between RFID enabled devices. The main focus is on encryptions algorithms.

To achieve this goal, a base of knowledge regarding NFC systems must be established. Therefore, at first, there will be an introduction to the RFID systems, followed by a technical overview for NFC systems and different operation modes.

2. Literature Review

NFC topic is still relatively new in the literature sense, and the idea in its basic form developed from the older RFID communication systems. The advancement in mobile computational power has facilitated the emergence of new application. Due to this reason, most of the literature deals with application development and how it utilizes NFC systems in different areas. Literature regarding NFC security is somewhat sparse and about one fifth of all the literature regarding NFC is related to security. This is mainly due to the nature of NFC systems, the close proximity provides a natural immunity against several attacks, and partially due to the fact that NFC systems is still new and not widely used in different areas communication and commerce.

Naser Hossein Motlagh details the NFC systems and the background from which it was derived in his Master level thesis from the University of Vaasa. NFC stands for Near Field Communication, and in definition it refers to short range wireless communication technique operating on 13.6 MHz with 14 KHz bandwidth. Initially developed by Sony and Philips and derived from RFID protocols, NFC devices are backward compatible with other RFID enabled devices. The main advantage of NFC systems is the fast detection and setup time. Hence it can be used to facilitate other forms of communication such as Bluetooth or WLAN connection without any manual configurations, by just simply bringing the two devices in close proximity to each other and the connection is set. Other applications include Peer-to-Peer mode to transfer data over the NFC connection, Reader/Writer mode in which an NFC enabled device can manipulate the content stored in passive RFID tags or smart cards, and Card Emulator mode, that allows the NFC enabled device to “mimic” a smart card so it could be read by an external smart card reader (i.e. debt card reader). (Hossein 2012)

The paper “NFC Devices: Security and Privacy” by Madlmayer, Langer, Kantner and Scharinger examines the issue of privacy and security, after a short introduction of NFC systems and its component. Several key Scenario Cases were introduced, based on them, threat model assumptions and suggestions were given, regardless of the actual feasibility of the threats. Also, they have defined the trust levels of the NFC systems components and how to improve them, what are the risks related to each component itself, and the connection to other components. The proposed counter measures include

dynamic ID with no ID based services, a manual control to switch on/off NFC, which includes a special mode (NFC flight mode). Other recommendations are related to the secure element in the NFC systems, which include authenticating the application index in the Secure element, managing the in device security, and finally integrating a security layer for the peer to peer communication link. The assumptions made in this paper are more focused on the integrity of the NFC system components, and they briefly outline the possible attacks that might occur during the communication between two devices. However, the attacks listed in this paper are generic and not intended for all wireless systems, since some of them are not a threat to NFC systems due to the connection nature of NFC systems (i.e. close proximity). (Madlmayer 2008)

Ernst Haselsteiner and Klemens Breitfuß from Philips Semiconductors published a paper during Workshop on RFID Security RFIDSec (2006), under the name "Security in Near Field Communication (NFC) Strengths and Weaknesses". In their paper, they briefly discussed the applications and operation modes, they also discussed the most common threats for any wireless communication system, and how feasible they were to NFC systems. Based on their research results they suggested solutions to mitigate them. In their paper they stated that the nature of NFC was highly secure but not complete. Also, they showed that the man in the middle attack was practically impossible, other attacks like eavesdropping, and data corruption/data modification/insertion were possible. These could be mitigated by using a secure channel. Using standard key agreement protocols or a specific key agreement that is designed especially for NFC systems would provide an improved level of security. However, their solutions require both ends of the communication system to be always listening which in turn results in more power consumption. Also, the asymmetrical keys agreements like Elliptic Curve Cryptography or RSA require a higher processing power, which might not be available in certain low cost NFC enabled devices. (Haselsteiner & Breitfuß 2006)

Gauthier Van Damme and Karel Wouters took a different approach with their paper "Practical Experiences with NFC Security on mobile Phones" by designing a Secure NFC Offline Payment Application, based on their analysis of different possible threats and attacks. Also, they demonstrated the complexity and the byproducts of designing an application where the security level and privacy integrity require the highest priority.

In their paper, they used two Nokia phones (6131 and 6212) running on Symbian S40. They analyze the Secure Element available in the device, and how it affects the transaction characteristics. Finally, they demonstrate the practical problems in the implementation. Most of the problems they encountered are related to the programming language used in the secure element, which increased the time of executing the cryptographic protocols, and limited the programmers from optimizing the application. (Van Damme & Wouters 2009)

Collin Mulliner presented a method of testing and analyzing the vulnerabilities and possible attacks on NFC-enabled mobile phones through the application of Fuzzing using NFC Tags. The NFC system itself was analyzed, along with the other components which can be controlled via NFC, including Telephony subsystem and the web browser. During the testing several attacks were possible, and a survey was made among several service providers to check the feasibility and the impact of these attacks. The testing device was again the Nokia 6313 running the Symbian S40 operating system. As discussed previously, the OS has several limitations that brings to the forefront vulnerabilities in the security and allows different exploits to occur. (Mulliner 2009)

There are several factors that affect the amount of publications regarding the security analysis of NFC systems. First, the lack of standardization makes testing, and developing solutions for security and privacy measures more complicated and less efficient. For example, Mulliner used the Nokia 6313 device running Symiban S40 and Van Damme & Wouters used the same device. Even though both papers were completed in the same year, the testing methods were different, and the focus/purpose of the research examined different points of view within the security aspect of NFC. The second factor refers to the variety of operating systems, the different processing power and the different programming languages. This factor hinders the researchers from reaching a general solution for a certain application mode. (Mulliner 2009), (Van Damme & Wouters 2009)

Most publications are focused on applications within the reader/writer communication mode (Proximity Coupling Device, PCD), mainly reading and writing on passive tags. However, the main focus of this research is to examine the different encryption algorithms used in NFC enabled devices regardless of the operation mode to assess the encryption/decryption time and power consumption in NFC enabled devices, as well as their immunity to brute force attacks.

3. Technical Overview

3.1. RFID Overview

RFID systems use Radio Frequencies for communication in order to identify the tagged objects. When a tagged object enters the read vicinity of the interrogator, the reader initializes the communication by sending a signal. The tag receives this signal and uses it as an energy source to send back the stored data. Tags can hold different types of data about the tagged object; these data can include the serial number, time stamps, and configurations and so on.

Based on the frequency used in the RFID systems (tags and interrogator design), the range of communication may vary in range. Low frequencies (LF) and high frequencies (HF) have short identification ranges, but low cost. Ultra High Frequencies (UHF) and microwave frequency band are used in long ranges and high data handling rate (Balanis 2005).

3.1.1. RFID components

A RFID system consists of three main components. A transponder (tag), a reader (interrogator) and a controller, as shown in Figure 2

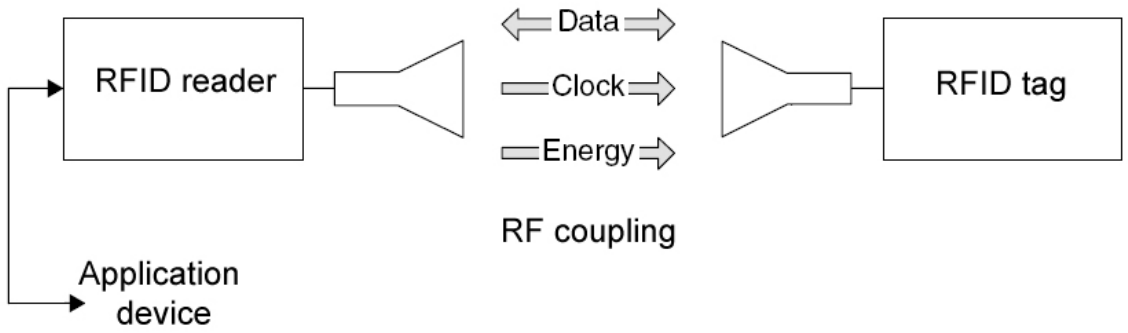

Figure 2. The Three Main Components of an RFID system.

The reader (interrogator) is a read/write device. It contains an RF module for transmitting and receiving signals, attached to the RF module is the control electronic module and an antenna. The Tag (Transponder) is a semiconductor chip attached to an

antenna. Each tag contains a unique serial identification number which facilitate the communication with the reader.

The RFID reader is connected to the workstation where a specific control application is running (e.g. computer), where software commands and database are stored. The RFID reader sends the data information alongside the Clock and energy (in case of passive tag) to be received by the tag.

3.1.2. Classification of RFID systems

The RFID systems are classified according to two parameters, one is energy source, and the other is the used frequency band.

Passive mode of communication is when the received signal is used to energize the transponder. That means that the tag does not include any independent energy source. Active mode is when the tag circuit includes an independent energy source. These two types can be divided by the frequencies used for communication, as it is seen in Figure 3.

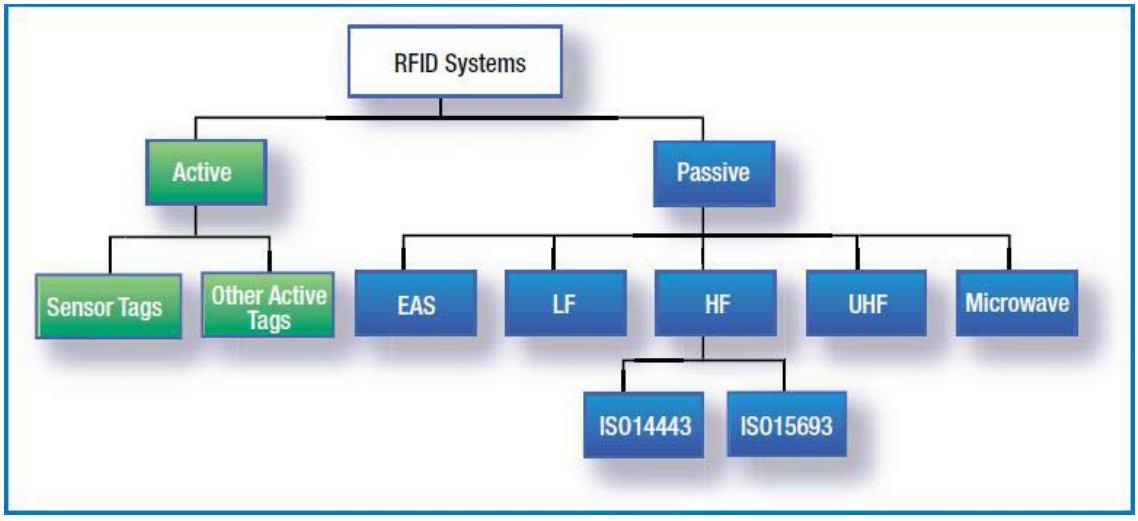

Figure 3. RFID System classifications (Atmel 2010: p45)

Passive tags are physically smaller and less expensive than active tags, due to the lack of a power source. Each mode of communication has its own pros and cons, and it is up to the system designer to choose which tag type is more suitable. Table 1 illustrates the difference between active and passive mode.

Table 1. Active and Passive RFID tags Comparison.

Features	Passive RFID	Active RFID
Power Source	External (Reader Provided)	Internal (Battery)
Tag Readability	Only within the area covered by the reader typically up to 3 meters	Can provide signal over an extended range, typically up to 100 meters
Energy	A passive tag is energized only when there is a reader present	An active tag is always energized
Magnetic Field Strength	High, since the tag draws power from the electromagnetic field provided by the reader	Low, since the tag emits signals using internal battery source
Shelf Life	Very high, in ideal case does not expire over the life time	Limited to about 5 years (The life of a battery)
Data Storage	Limited data storage, typically 128 bytes	Can store larger amount of data
Size	Small	Depends on the battery size
Cost	Cheap	Expensive

3.1.3. RFID coupling mechanism

Based on the application and the distance separating the reader and tag, several coupling types and mechanism exists, to each method its own different features. The three main mechanism of coupling which allows the reader and the tag to communicate are backscatter coupling, capacitive coupling and inductive coupling.

RFID Backscatter coupling: The reader propagates radio signals outside the near field region. The tag receives the signal and uses it to energize the embedded chip and reflects the rest of the signal as data. This type of coupling is used for medium distance between the reader and the tag. Figure 4 demonstrates backscatter coupling (Cisco Wi-Fi Location-Based Services 4.1 Design 2008:158).

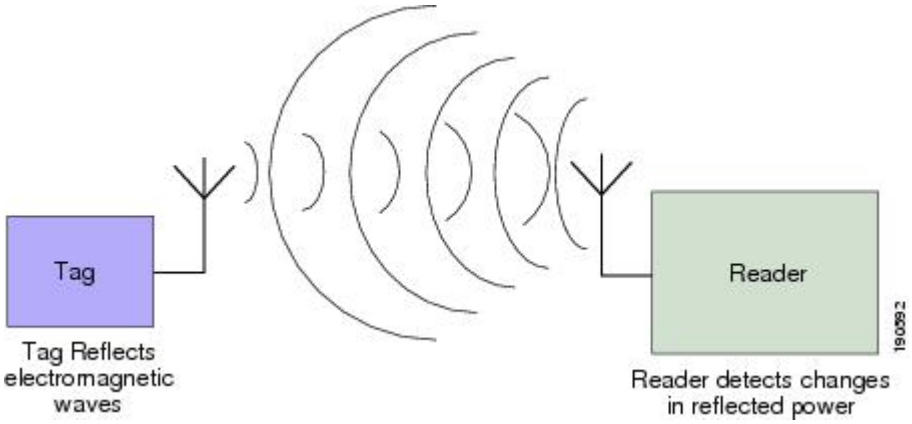

Figure 4. Backscatter Coupling.

RFID capacitive coupling: as the name implies, the coupling is done by utilizing the capacitive effect, this technique is most suitable for short range communication. The capacitance between the reader and tag provides a conducting capacitor, through which a signal can be transmitted, although an earth return is required. The AC signal generated by the reader is receive and rectified within the RFID tag and used to power the circuits within the tag. Again the data is returned to the RFID reader by modulating the load. Figure 5 is an illustration of capacitive coupling (Gorferay Card Service Co. Ltd)

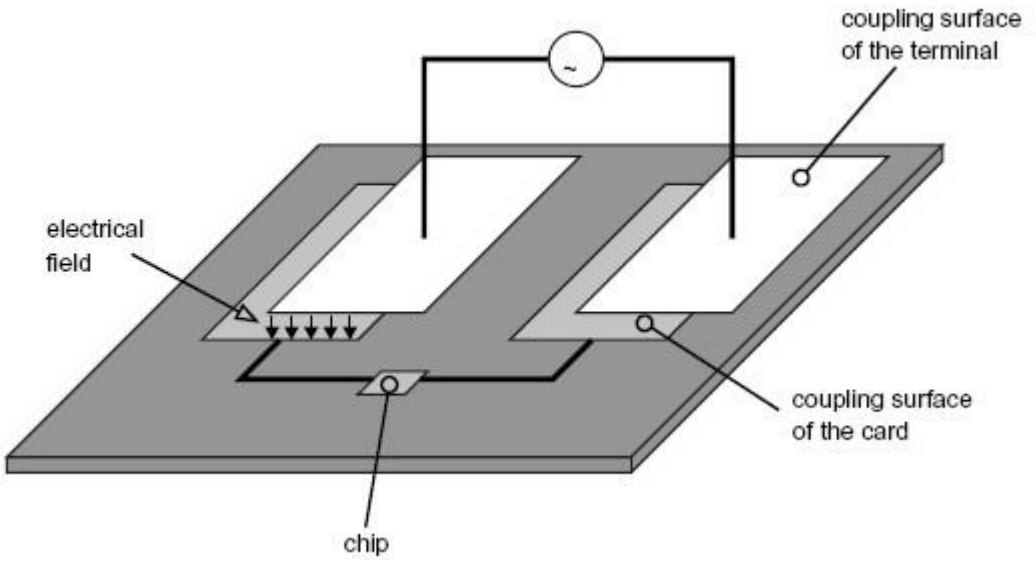

Figure 5. Capacitive Coupling.

RFID inductive coupling: this technique achieves coupling by utilizing mutual inductance between the reader and the tag circuits. Inductive coupling is also suitable for short range coupling. It can be used for a slightly longer range than capacitive tags. RFID inductive coupling requires that both the tag and the reader to have induction or "antenna" coils. When the tag is placed close enough to the reader, the field from the reader's coil will couple to the field from the tag's coil. A voltage will be induced in the tag that will be rectified and used to power the tag circuitry. Inductive Coupling is illustrated in Figure 6 (Glover 2006).

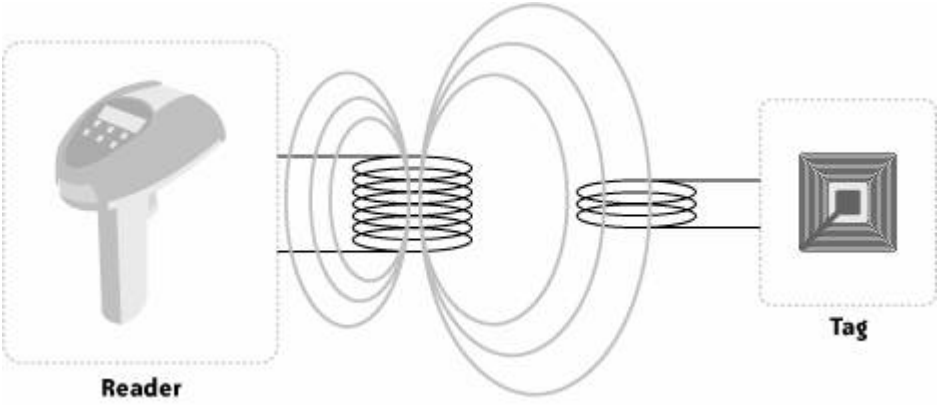

Figure 6. Inductive Coupling.

3.1.4. RFID frequency bands

As stated earlier, the frequency bands at which the RFID system operates defines its range and classification. RFID systems operate in the unlicensed radio frequency bands known as ISM (Industrial, Scientific and Medical) but the precise frequencies which are defined for RFID may vary depending on the regulations in different countries. These frequency bands are listed in Table 2.

Table 2. Common RFID Operating Frequencies and Features

Band	LF (Low Frequency)	HF (High Frequency)	UHF (Ultra High Frequency)	Microwave
Frequency	30 – 300kHz	3 – 30MHz	300MHz – 3GHz	2 – 30 GHz
Typical RFID Frequency	125-134-KHz	135.6MHz	433MHz/865-956 MHz	2.45Ghz
Approximate Read Range	Less than 0.5 meter	Up to 1.5 meter	Up to 100 meter/0.5-5 meter respectively	Up to 10 meters
Data Transfer Rate	Less than 1kbps	Around 25kbps	30kbps	Up to 100kbps
Characteristic	Short range low data transfer rate, penetrates water but not metals	Higher ranges, reasonable data rate (similar to GSM phone), penetrates water but not metals	Long range, high data transfer rate, concurrent read of <100 items, cannot penetrate water or metals	Long range, high data transfer rate, cannot penetrate water or metals
Applications	Animal ID, car immobilisers	Smart labels, contactless travel cards, access & security	Specialist animal tracking, logistics	Moving vehicle tolls

LF and HF RFID systems are used for near field communication and the inductive coupling mechanism. UHF and higher frequencies RFID systems are used for far field communication with the backscattering coupling.

3.1.5. RFID standards

None of the applied standards in RFID systems is universal. These standards may be categorized into four levels of international, national, industry and association level. These standards cover four key areas of RFID application:

- Air interface standards which are used for basic tag to reader communication
- Data content and encoding i.e. the format of the codes used in tags
- Conformance which means testing the RFID system
- Interoperability between applications and RFID system

There are several standards that define the development of RFID technologies such as:

- International Organization of Standardization (ISO)
- Electronic Product Code (EPC)
- European Telecommunication Standards Institute (ETSI)
- Federal Communication Commission (FCC)

Each of the standardization organization mentioned above defines a set of standards for different RFID applications while ISO supports the required standard for RFID frequencies under series of ISO 18000 which are known as Air Interface Family.

3.2. NFC overview

Sony and Philips were the first to develop and initiate the NFC technology. Derived from RFID and with new added interface and protocols, NFC devices are still compatible with RFID technology.

Several differences separate NFC devices from RFID devices, bidirectional data transfer is one of them, also peer to peer communication, where in passive mode only one device produces the required radio frequency for the communication and the other device utilizes the load modulation for data transmission (Hossein 2012). The device that starts the communication is called “initiator” and the receiver is called “Target”.

3.2.1. NFC coding and bit representation

NFC utilizes Manchester coding and modified miller coding schemes to encode the commands, responses and data transaction. Manchester coding is one of the most common coding techniques applied nowadays, due to its many characteristics. An important characteristic is called “self-clocking”, that allows it to be used in inductive and capacitive coupling. Also, the clock signal can be recovered from the encoded data. Another property is bandwidth efficiency. Less bandwidth is needed to achieve a certain data rate, although it may be vulnerable to frequency errors and jitter in the sender/receiver reference clock.

Manchester code ensures frequent line voltage transitions (in the middle of each bit duration), directly proportional to the clock rate; this helps clock recovery. Also, it means that this coding method has two possible transitions at the middle of a symbol period. Manchester coding is shown in both Table 3 and Figure 7.

Table 3. Machester coding.

Original Data	Clock	Manchester Value
0	0	0
0	1	1
1	0	1
1	1	0

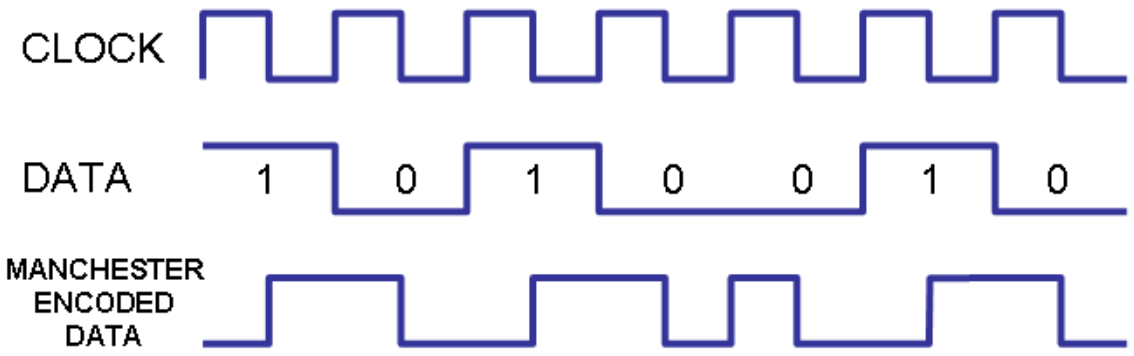

Figure 7. Manchester Coding.

Modified Miller Coding is another coding method used in NFC enabled devices. Ones and zeros are defined by the position of the pulse during one bit duration.

For logic one, the transition from HIGH to LOW happens in the middle of the bit duration, the pulse occurs in the second half of the bit period. For logic zero, a pulse shall occur at the beginning of the bit period. But when the ZERO bit is preceded by the ONE bit, no pulse shall occur during this ZERO, as it is shown in Figure 8 (Standard ECMA-373 2012).

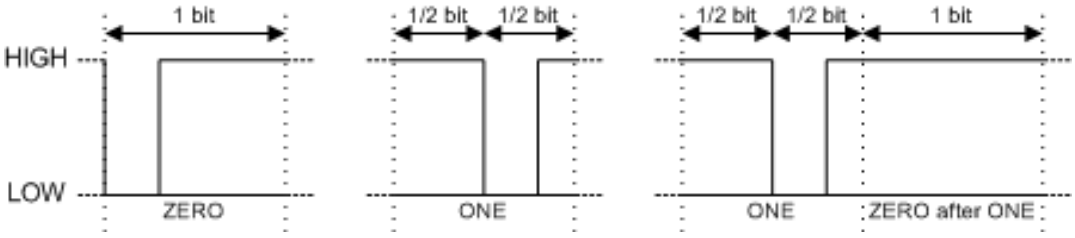

Figure 8. Modified Miller Coding.

3.2.2. NFC characteristics (frequency and data rates)

NFC is designed for short range communications - maximum theoretical range of 20 cm but practically around 4 cm - operating in the globally available unlicensed band of 13.56 MHz with a bandwidth of 14 KHz. It supports data rates of 106 kbps, 212 kbps, and 424kbps. Higher bit rates are possible depending on future development. NFC uses half duplex channel (same frequency is used for sending and receiving data). To prevent collision, NFC devices utilize Carrier Sense Multiple Access (CSMA) protocol. NFC characteristics are presented in Table 4.

Table 4. NFC Characteristic

Rate	Speed	Active Device	Passive Device	Description
$f_{CLK}/128$	106	Mod. Miller, 100% ASK	Man, 10% ASK	The sent signal is the product of an AND operation between the coded data using modified miller bit and the clock signal. The length of the modified miller bit coded pulse varies between 7 and 45 cycles long.
$f_{CLK}/64$	212	Man, 10% ASK	Man, 10% ASK	The sent signal is the product of an XOR operation between the data coded using Manchester Bit-Coding and the Clock Signal.
$f_{CLK}/32$	424	Man, 10% ASK	Man, 10% ASK	The same as 212 Kbps Data Rate.

3.2.3. Communication modes

As in RFID devices, NFC has two modes of communication, depending on the device itself. If the device can generate its own radio frequency field, then it is called active, otherwise it is passive. Active communication mode occurs when two active devices communicate with each other, passive mode requires one active device.

The International Standard specifies requirements for modulation, bit rates and bit coding. In addition, it specifies requirements for the start of communication, the end of communication, the bit and byte representation, the framing and error detection, the single device detection, the protocol and parameter selection, and the data exchange and de-selection of Near Field Communication Interface and Protocol (NFCIP-1) devices (ISO/IEC 18092 2013(E)). The transaction of commands, responses, and data starts with the devices initialisation and end with the device de-selection, in alternating or half duplex channel.

The devices are capable of establishing the transaction using 106/kbps, 212/kbps, or 424/kbps. The initiator selects one of those bit rates, the bit rate may be changed during

the transaction using certain commands and protocols. However, the mode of the communication (active or passive) cannot be changed.

In the Active communication mode, both the initiator and the target use their own RF field to communicate. The initiator starts the NFC protocol (NFCIP-1) transaction. The target responds to an initiator command in the active communication mode by modulating its own RF field. (ISO/IEC 18092 2013(E))

- 106 Kbps data transmission (low rate)
  The initial speed of any transaction is always the lowest rate possible of 106 Kbps. At this rate the initiator apply 100% ASK modulation to generate the required pulse (Hossein 2012). The coding technique used for this rate is the modified miller coding. The serial data transmission systems send the least significant byte first (LSB) followed by the rest of the data.

- 212/424 Kbps data transmission (high rate)
  The modulation scheme used for these rates is still ASK. However, the index is different, for these rates the modulation indexes are 8% and 30% of the operating field. The serial data transmission systems send the most significant byte first (MSB) followed by the rest of the data, also the reserve polarity in the amplitude of the Manchester symbols is allowed. The target shall respond with the same load modulation scheme but the bit duration must be altered to the actual bit rate (Ecma 2004).

In the passive communication mode, the initiator generates the RF field and starts the transaction. The target responds to the initiator command in the passive communication mode by modulating the initiators' RF field, which is referred to as load modulation (ISO/IEC 18092 2013(E)).

The communication between the initiator and the target follows the same procedures as in the active mode (since the initiator generates its own RF field). The response from the target to the initiator; however, it follows different procedures.

The target responds by load modulation which generates a sub carrier with frequency of ($f_s=f_{CLK}/16$). The load modulated signal's amplitude has to be greater than the existing

magnetic field strength threshold. The signal is coded using Manchester coding, and the bytes are encoded with LSB first for the lower data rate (106 kbps) and MSB first for the high bit rates (212 and 424 kbps).

3.2.4. NFC protocols

All NFC enabled devices are in target state by default, to preserve power and not to disturb any on-going communication in the vicinity. All NFC devices will not generate an RF field. The device changes its state upon a request from the control unit (program on the mobile device or computer) to be an initiator, the program determines the communication mode and the data rate. The initiator use collision avoidance protocol to detect any existing radio field before initiating communication with the target, the target device waits for the initiator field to respond.

In collision avoidance procedure, the initiator “listens” and tries to detect if there is any other RF field. After a certain amount of time known as the guard time, the device can initiate communication if no RF field is detected in order to prevent disturbing other NFC devices. If two or more targets answer the initiator’s field simultaneously then a collision will be detected and simply the frames will be discarded. Collision avoidance is defined to avoid similar issues and to minimize disturbance with other on-going communications at the same frequency, the initiator should not generate any RF field until the time of the existing field is terminated (ETSI 2003). Figure 9 describes the general initialization and single device detection (SDD) for the active and passive communication mode at different transfer speeds.

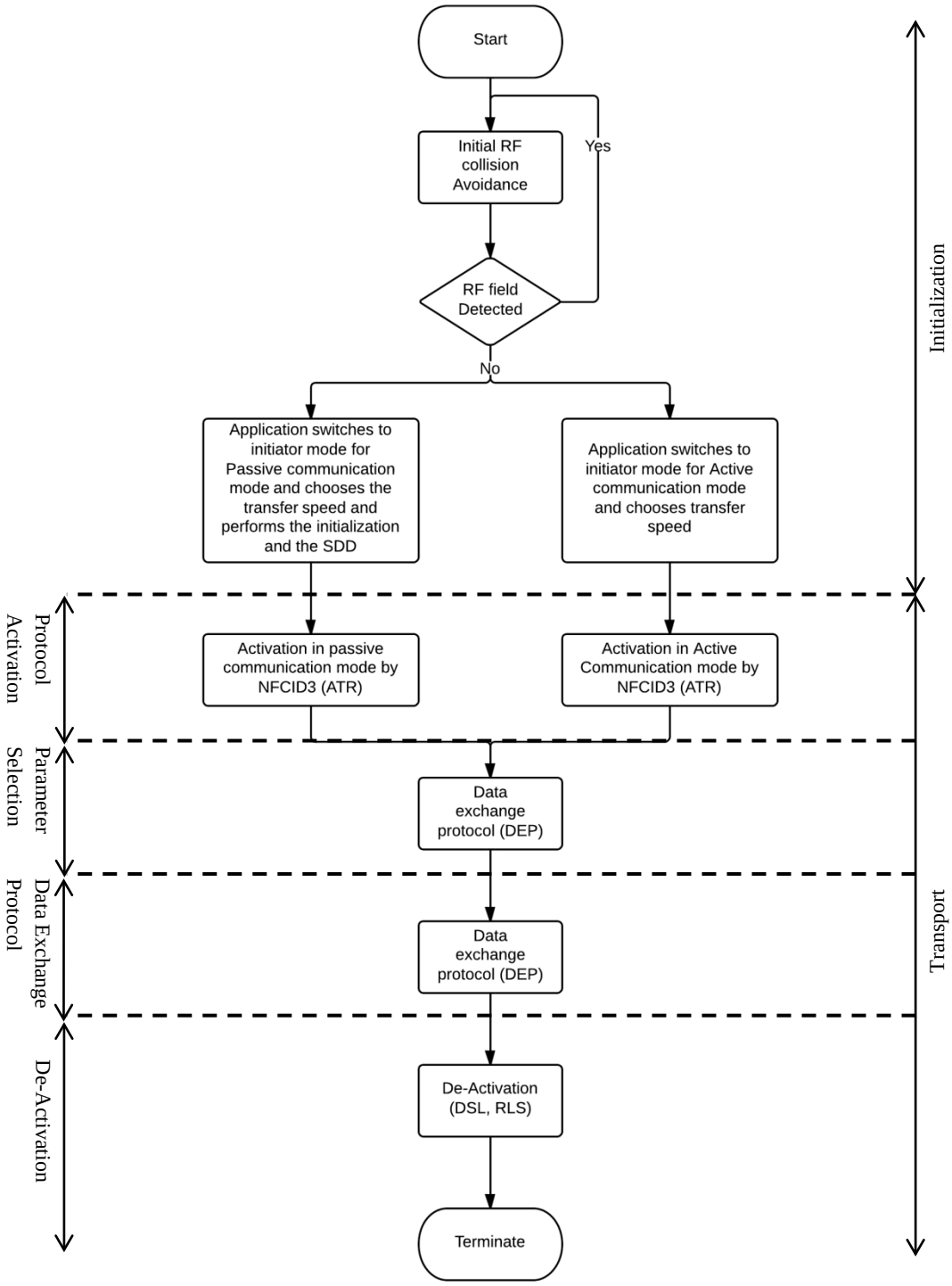

Figure 9. Initialization and single device detection (Near Field Communication -Interface and Protocol (NFCIP-1))

3.2.5. Frame formatting

The data exchanged in NFC vary from one to another depending on the source (initiator or target) and on the mode of communication (active or passive). For each type, the data is grouped in what is called frames. The basic frame format for the initiator and target is shown in the Figure 10.

Start of communication (START)	Information	End of communication (END)

Figure 10. Initiator and target general frame format.

3.2.6. Passive communication mode

In initialization and single device detection for 106 kbps in passive mode, two frame formats are used at the 106kbps data rate. The short format which consists of 7 bits is used in the initialization step. The initialization short frame is presented in Figure 11.

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit5	Bit 6	
START	Commands							END

Figure 11. Initialization step Short frame format at 106kbps.

After the initialization step, standard frame format is used for data transfer at 106 kbps, the standard frame format is shown in the Figure 12.

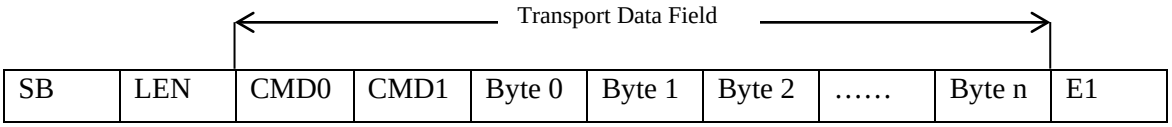

Figure 12. Standard frame format at 106kbps.

SB byte stands for “Start Byte” in the data exchange protocol at *fc*/128 and is set to 0xF0. LEN defines the length of the transport data field plus 1, and E1 declares the end of transmission.

Initialization and single device detection for 212 & 424 kbps in passive communication mode starts once the carrier frequency field is detected. The communication starts with

the minimum preamble sequence of 48 bits Manchester encoded ZERO. The end of the communication is declared in the length field of the frame.

The frame format consists of Preamble, SYNC, Length, Payload, and CRC fields. They are defined in Figure 13 (Near Field Communication -Interface and Protocol (NFCIP-1)).

Preamble	SYNC	Length	Payload	CRC

Figure 13. Standard frame format.

Preamble: 48 bits minimum all logical ZEROs. Its function is to signal the beginning of communication.

SYNC: bytes. The 1st byte of the SYNC is ‘B2’ and the 2nd byte is ‘4D’. This frame is to synchronize the field frequency with target clock frequency.

Length: an 8-bit field and it is set equal to the number of bytes to be transmitted in Payload plus 1. The range of the Length is between 2 and 255. Other settings are reserved for future use “RFU”.

Payload: consists of N number of bytes of data.

CRC: Are the cyclic redundancy check bits.

Basic method of single device detection (SDD) at 212 and 424 kbps is the Time Slot method. The initiator sends polling requests, the target responds at random in each time slot. The target’s identification number for passive communication mode (NFCID2) will be read by the initiator. Once the NFCID2 data from the target is obtained, in the operating field, the initiator can communicate with multiple targets.

In order to find a target, the initiator sends a Polling Request frame, which is a standard frame format with predefined parameters and values such as the synchronization bits and the payload.

Similar to the polling request, the polling request response is also a standard frame with some predefined parameters. Instead of the regular values in the payload, the polling

request response payload consists of one byte set to ‘01’, the NFCID2 data, and the Pad information which shall be ignored for data interchange. Other differences in the length segment and CRC bits. Synchronization bits and preamble are the same in the polling request and the polling request response.

3.2.7. Passive communication mode activation flow

The initiator performs the initial RF collision avoidance sequence. After that, it performs the initialization and single device detection for passive communication mode at a chosen transfer speed. The target checks the protocol at different transfer speed according to the attribute request. If the attribute request is not supported by the target, the target may fall back to the initialization and single device detection. The attribute response is sent by the target as an answer to the attribute request sent by the initiator. The target shall only answer if the request is received directly after the selection. If the target supports any changeable parameter in the attribute request, a parameter selection request may be used by the initiator as the next command after receiving the attribute request to change parameters. The target response to the initiator request, but if it does not support the change of parameters, then the target does not need to compliment on the request of the initiator. The data exchange transport protocol will be active now and the transparent data will be sent.

The Initiator activation sequence for a target in the passive communication mode is seen in Figure 14 (ISO/IEC 18092 2013)

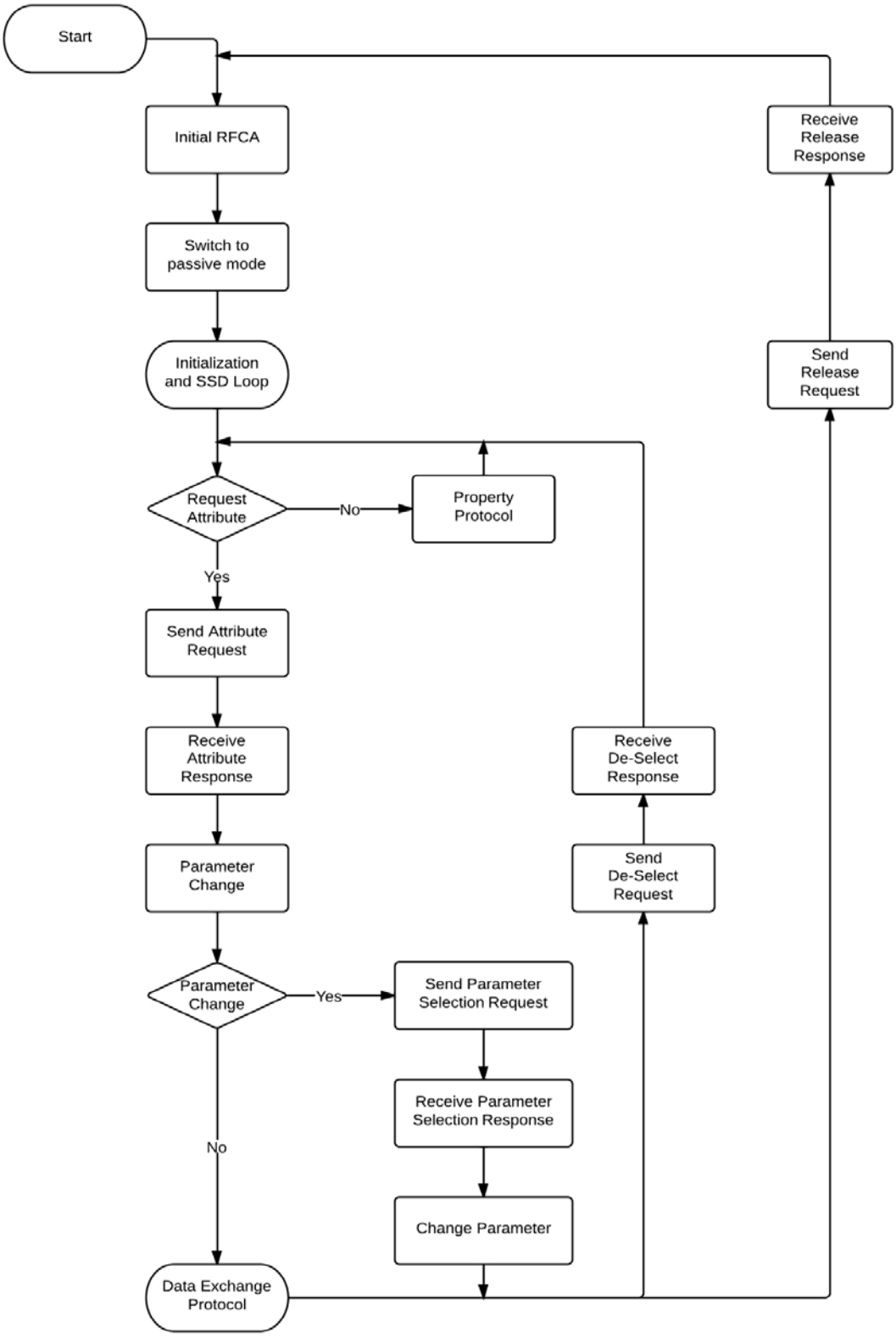

Figure 14 Activation protocol in Passive communication mode

2.2.8. Active communication mode

In the initialization stage, the application controls the device and switches to initiator for active communication mode, choosing one of the three data rates. The active communication mode RF with collision avoidance works in the following way. First the initiator performs an initial RF collision avoidance, by first sending the “attribute request” command in the active communication mode at the selected transfer speed. Second, the initiator switches off the RF field in order to allow the target to perform the response RF collision avoidance, the target sends “attribute response” command in the same speed chosen by the initiator and switches off the RF field. Then the initiator performs the response RF collision avoidance. After that, the initiator sends “the parameter selection request” in order to change parameter or sends the “data exchange protocol” to start the data exchange protocol.

In case of two targets or more in the field, the initiator chooses only one target and ignores the others. And if the two targets answer in the exact same time period, the initiator will discard targets as it will detect a collision and will resend the “attribute request” as described earlier.

The transport protocol is handled in three stages: first is the Activation of the protocol, which includes the Request for Attributes and the Parameter Selection. The second stage is the data exchange protocol, and finally the deactivation of the protocol including the Deselect and the Release.

The position of the data field varies depending on the data transfer rate, for each data rate the frame format is different. The transport data rate field contains the mandatory command bytes and the data bytes.

3.2.9. Active communication mode Activation flow

In active communication mode, the initiator at first performs an initial RF Collision avoidance sequence, and then it will switch to active communication mode and select the transfer speed. After that, the initiator sends an attribute request. The target in the operating field will send a response to the initiator request, in case more than one target responded, the initiator will detect a collision and the attribute request will be re-sent. Once one target responds successfully, the device is selected and the initiator may send a parameter selection request depending on the attribute response sent by the target, otherwise if the target does not support any changeable parameter, it does not need to complement on the initiator request. The Initiator activation sequence for a target in the active communication mode is shown in Figure 15 (ISO/IEC 18092 2013).

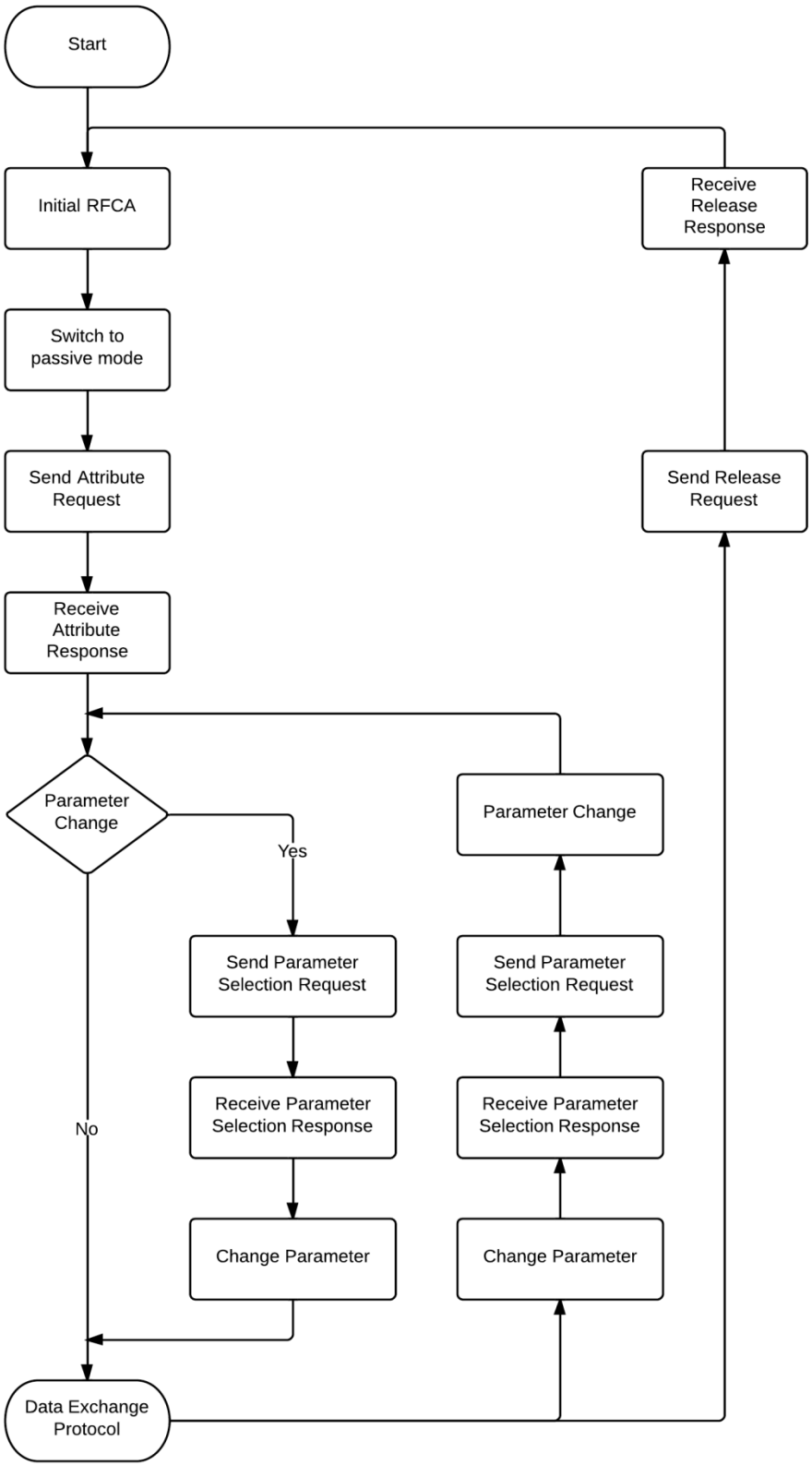

Figure 15. Activation protocol in Active communication mode

3.2.10. NFC communication modes

NFC Interface and Protocol (NFCIP-1) is defined by ECMA-340 and ISO/IEC-18092, which presents the operation modes of NFC devices – passive or active – and the transfer speed rates. This protocol alongside the ISO14443 (Contactless card standard) and ISO 15693 (Vicinity cards) have been expanded to the Near Field Communication Interface and Protocol-2 (NFCIP-2) to form The EMCA-352 (ISO/IEC-21481) standard. In this way any NFCIP-2 compliant device is compatible with all devices communicating on 13.4MHz. These standards led to three basics communication modes, Peer-to-peer, Read/Write, and Card Emulation mode. (Van Damme and Wouters 2009)

Peer to Peer (Near Field Communication, NFC) is the classic mode of communication. By holding to active devices in each other's operating range, a connection is established on the link level, allowing data to transfer at rate up to 424Kbps. This mode follows the Master/Slave principle of communication.

In Read/Write Reader/Writer Mode (Proximity Coupling Device, PCD), when passive NFC compatible tag or a passive smart card is in the range of an active NFC capable device, the active device switches to be an initiator. The tag is energized, the initiator is able to read data from the tag, and write data as well. Depending on the running application, data transfer rate depends on the tags properties; 106Kbps is supported in this mode.

Card Emulator (Proximity Inductive Coupling Card, PICC) mode is at which the NFC enabled device acts as smart card following ISO 14443 Protocols. An external NFC reader will "see" the device as a smart card, and the reader is able to read information from device.

Figure 16 illustrates the differences in the communication modes.

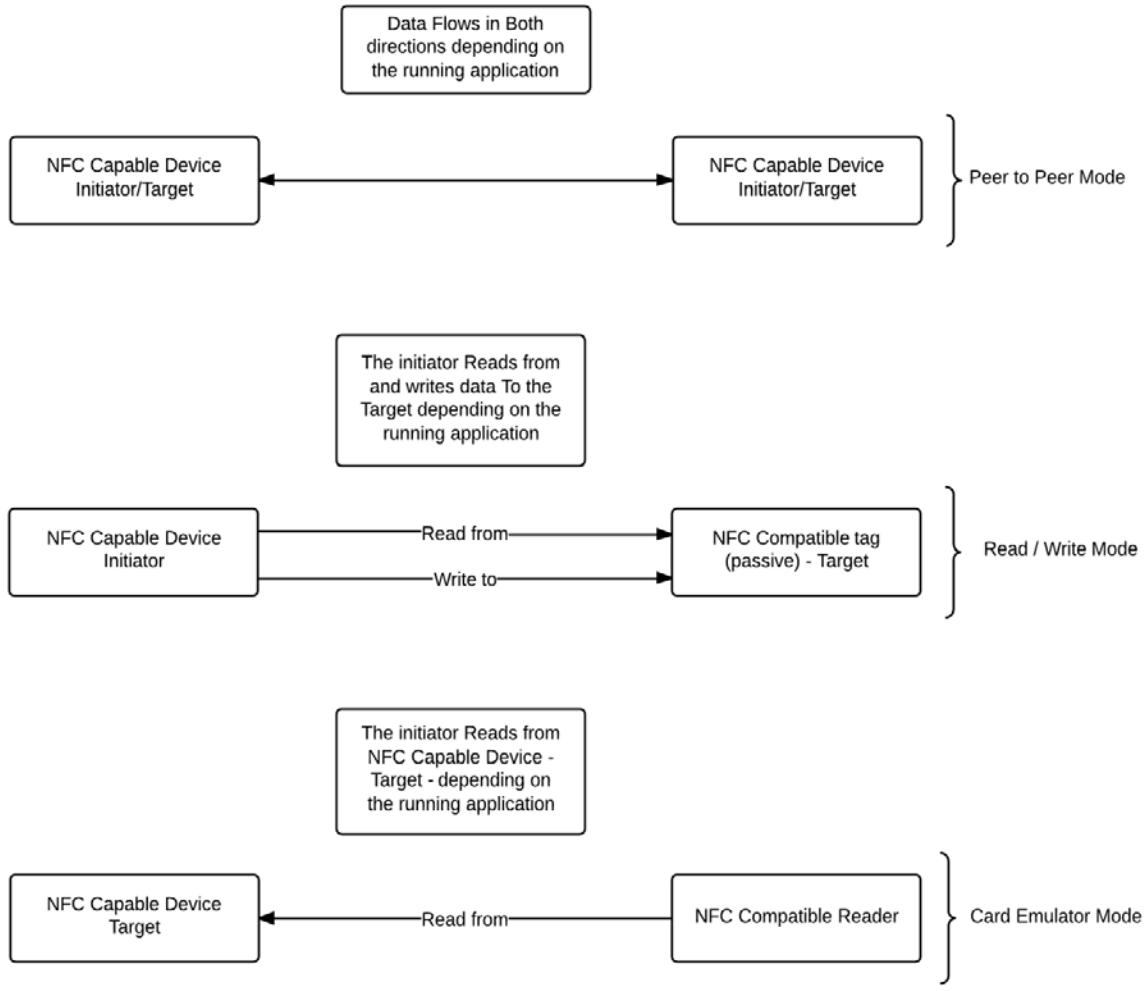

Figure 16. NFC Communication Modes

3.3. Smart Cards

The vast variety of applications served by smart cards makes categorizing them a hard task. However, there are two properties that help separating the smart cards into major categories; these are processing power, and connection type.

Memory cards are the first type of smart cards and they were used for telephone applications, as prepaid cards. The value was stored electronically in the chip, and it was decreased every time the card is used. Security measures were added to the cards in this application to eliminate manipulation and any other exploits. These pre-paid cards were used as a one-time use only, once the balance stored in the cards is depleted, the cards cannot be recharged and are no longer useful.

The main advantage of this type of cards is the simplicity of its implementation, which makes the cost to manufacture them very low. However, since they cannot be reused again once empty, they must be discarded. Memory cards have limited functionality due to their simplicity, protecting the stored data is possible, but they are mainly used in application where low cost is the primary concern.

The other type is processor cards. As the name implies, these cards contain embedded processors. The embedded processor and memory in the cards make them flexible to serve more applications due to their ability to store secret keys securely and execute modern cryptographic algorithms. The only restriction on the functionality of the card is the available resources – memory and computing power – otherwise, the developer can program the card as needed. The most wide spread application of processor cards is their use as access medium for the European Digital Mobile Telephone system (GSM), partially because of the high security level that they can achieve while accessing the mobile telephone network. The other main reason is that they have provided new possibilities for both the network operators and service providers to sell their products separately. Other applications of processor cards mainly demand a high level of security, such as personal identification in some restricted areas or on some computers or equipment, banking information for credit or debit cards are also another field where processor cards are used.

The ability to perform cryptographic algorithms and the ability to securely store confidential data are the essential advantages of processor cards makes use processor cards in new application is inevitable. The steady decline of production cost due to mass production and ongoing technological progress helps further expands the use of smart cards.

The previous two types of smart cards are based on the processing power abilities of the card. Another property which can be used to categorize smart cards is the connection type. This includes contact and contactless cards

Contact Cards need to have a physical connection with the terminal. This connection is done via the connection plate in the card. Through this connection the card is energized and can transfer data from and to the terminal. Contact cards can be memory cards or processor cards, depending on the application and where the card is used.

Contactless cards do not need a physical connection with the terminal. The connection between contactless cards and the terminal is established though an RF link.

Inexpensive, mature, and mass produced contactless cards are available nowadays in both memory cards and processor cards forms. This is the result of the rapid progress in integrated circuit technology and the dramatic decrease in power consumption, allowing the energy and data to be transferred to and from the card without any electrical contact. Contactless cards range of operation is limited to few centimeters for power conservation reasons, it is possible to extend this range up to a meter away from the terminal, however the close range of operation increases the contactless cards security level. An attacker can read the content of the card and might even manipulate it without the knowledge of the card owner, if the range was relatively longer.

Dual interface cards are considered to be a possible solution to this particular problem. However, operation over a long distance should be prevented due to the high risk of security breach. Contactless smart cards are suitable for applications that demand quick identification, such as access control, public transportations, airline tickets and luggage identification and many others. (Rankl & Effing 2010: p9)

3.3.1. Physical Properties

The smart cards physical properties determines the cards format, material, placement of different components such as magnetic strips, chip module and its connections and many others. The card can also include embossing, holograms and other features.

The application where the smart card is used dictates many of its physical attributes. For example, if the operation environment is subject to high ambient temperatures, then both the smart card's body as well as the embedded microcontroller, must meet all the relevant requirements, individually and collectively.

The most notable property of the smart card is its format. This format is standardized by ISO7810 also known as “ID-1 Format”, as shown in the Figure 17. (ISO 7810)

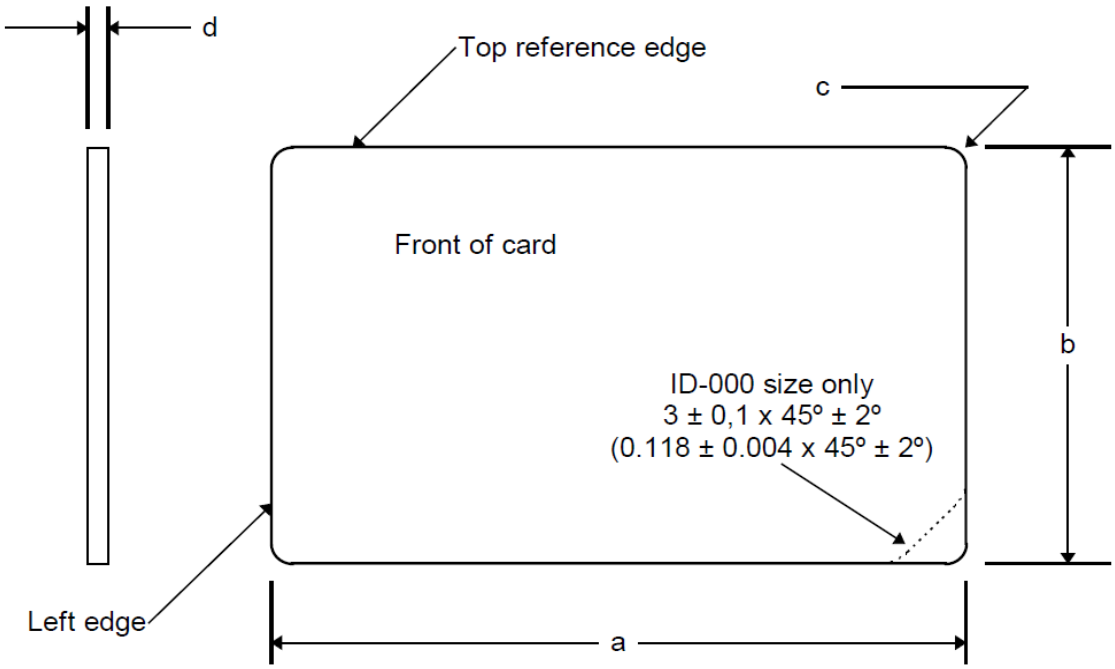

Figure 17. The ID-1 format as specified in ISO 7810.

However, this standard was issued in 1985, and at that time the idea of adding an integrated circuit to a card was not available. With the large variety of cards, each card has distinct dimensions to suit a particular application, it is not easy to determine whether a card is an ID-1 smart card or not. Besides having an integrated chip, an easy way to identify a smart card is to measure the card’s thickness - contactless smart cards

may not have a visible circuit on their surface - if its 0.76 mm then the card is a smart card (in reference to ISO 7810 Standards).

Other factors related to the card body, construction and material are determined by the designated function of that particular card. The mechanical attributes are also considered when manufacturing a card, depending on the environment at which the card is expected to operate. In order to protect the card from getting damaged, several properties must be inspected to ensure a high level of quality. These tests include mechanical robustness of the card, temperature resistance, electrostatic discharge, electromagnetic susceptibility, and many others.

3.3.2. Smart Card Micro Controllers

The key component of any smart card is the embedded microcontroller. It controls, initiates, and monitors all of the card's electrical activities. The smart cards microcontrollers have the setup of a full computer, they have a processor, memory (RAM, ROM EEPROM), and input and output interface to external devices. This hardware configuration must be governed by a suitably adapted and configured operating system which is tailored to fit different applications such as payments or telecommunication. Figure 18 illustrates the basic building blocks of a microcontroller.

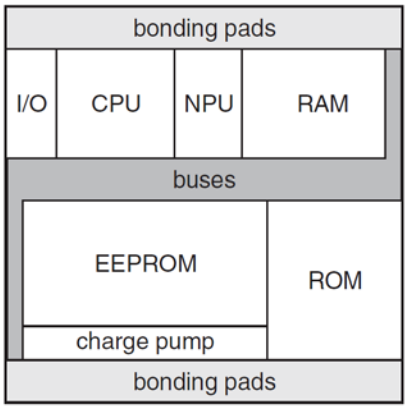

Figure 18. possible arrangement for essential functional components of a simple smart card microcontroller on the silicon die

The microcontrollers used in smart cards are not standard, widely available units, they are specially designed to serve a particular purpose. Some of the components from which the microcontrollers are made of however are already developed items for other ends, for example, the processors used in smart cards are not special designs, but instead proven components that have been used in other areas for a long time. Reasons behind this approach beside the high expense needed, is the complexity of developing a brand new processor with no suitable libraries and no available operating systems. Also, the processor in the microcontroller is not a special design, in addition to the stated reasons, the processor is the most important part of the microcontroller, and that's why it must be very reliable. Older processors which have been tested and proven reliable are used. (Rankl & Effing 2010: p73)

Several factors must be taken into consideration in the development of microcontrollers, to name a few:

Manufacturing costs, the structure width and the area of the microcontroller chip on the silicon wafer are the most critical factors regarding the manufacturing cost. Constant efforts are made to minimize the cost by reducing the chip area, also omitting any unnecessary additional components that occupy space without any useful functionality.

Functionality, this reason is closely related to the previous one. The area of the chip is very limited, and therefore only the components that provide the needed functionalities are included along the essential blocks of the microcontroller, anything else is omitted.

Security, smart cards provide tamper-proof storage of user and account identity. Smart cards also provide vital components of system security for the exchange of data throughout various types of networks. They protect against a full range of security threats, from careless storage of user passwords to sophisticated system hacks. Multifunction cards can also serve as network system access and store values and other data. (Smart card security basics, Cardlogix publication 2009)

Chip area is not only important from the cost point of view, but also from quality reasons, larger chips are more prone to be mechanically damaged. Smart cards are meant to be used in various situation hence, it is important that the card can handle

mechanical stress, most importantly the chip must handle a certain degree of mechanical stress, the finest hairline crack in the chip is sufficient to render the smart card useless.

In order to analyze the microcontrollers used in smart cards, it is necessary to analyze the components from which it is made. This include the processor, memory types, input/output interface and any other additional items used for a particular functionality

A typical microcontroller used in smart card application is an 8-bit processor with complex instruction set computer architecture (CISC). It has a memory capacity of 50-100 KB. And it is used without any significant limitations, which means it needs several clock cycles to execute each machine instruction and usually has a very large instruction set.

However 8-bit processors are not the only type of processors used in smart cards microcontrollers. To handle more complex tasks, 16-bit processors are used. They are based on architecture similar to RSIC architecture (‘RISC’ stands for ‘reduced-instruction-set computer’).

More recently, the highest performance possible can be attained from using 32-bit processors in smart cards microcontrollers. The direction of the development is heading towards 32-bits processors, due to the need for improved performance and for managing larger memories. The key selection criteria for processors include code density, power consumption, and resistance to attacks.

8-bits processors provide a solid basis for inexpensive microcontrollers at the lower end of the performance scale. Their power consumption and the area they occupy on the die are smaller compared to the 32-bits processor. However, 32-bits processors are needed for the more demanding application, for the processing power they can provide, with their broad bus structures and more elaborate internal components. (Rankl & Effing 2010: p82)

The other major component that affects the performance of the microcontroller is the memory. Different memory types are used in the microcontroller for various tasks, some of which are not essentials but to provide certain functionality. Different types of memory are illustrated in Figure 19.

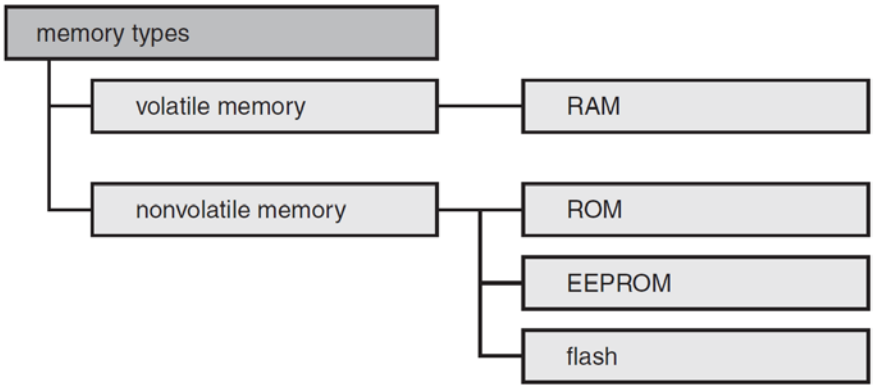

Figure 19. Memory types used in smart card microcontrollers. PROM and EPROM are normally not used in modern microcontrollers. FRAM is presently used relatively rarely in smart cards

The main two types of memory are separated by the way they maintain the data stored on them. Volatile memory requires power to maintain the stored information. It retains its contents while powered, but when power is interrupted stored data is immediately lost. Non-Volatile Memory on the other hand maintains its content even when unpowered.

RAM is an abbreviation for (Random-Access Memory) and it is a volatile memory. The number of accesses to the memory content is unlimited, the content can be modified by the running program as much as needed, as long the supply voltage is uninterrupted, the content of the memory will be intact.

ROM stands for Read-Only Memory. This type of memory can only be read and cannot be written, supply voltage is not needed to retain the data. A smart card's ROM contains most of the operating system routines, as well as various test and diagnostic functions. These programs are built into the chip by its manufacturer when it is fabricated.

EEPROM (Electrically Erasable Read-Only Memory) is technically more complex than ROM or RAM, it is used to store all the programs and data that need to be modified or deleted at some time. Its function resembles the hard drive function in a computer, since it retains data in the absence of power and the data can be altered as necessary.

The main goal is to minimize the required RAM and EEPROM in order to save space on the chip, since the RAM and EEPROM requires more space. As a rule of thumb, a RAM cell occupies about four times as much space as an EEPROM cell, which in turn

occupies four times as much space as a ROM cell. Other types of memory might be used to provide a certain function, such as flash EEPROM “Flash memory” or FRAM “Ferroelectric Random-Access Memory”. (Rankl & Effing 2010: p83)

Supplementary hardware are added in order to meet some requirements which the standard components cannot meet using only software. These requirements might not be fundamental tasks that are not possible to be done using software, but enhancement to improve the execution time and the power consumption. Some of these additional features include communication with a USB interface, Communication with MMC “MultiMedia Card”, the USB hardware enables the higher-level software layer to select a large number of configuration settings for the driver software. Like USB, MMC entails a large amount of additional hardware in the smart card microcontroller.

Communication with Single Wire Protocol (SWP). This is used for communication between a SIM and an NFC controller in a mobile telephone.

Additional components include a Timer, CRC “Cyclic Redundancy Check” Calculation Unit, a Random Number Generator, for security reasons all modern smart card microcontrollers have hardware random number generators.

In performance demanding application, clock generator and clock multiplier are used. The processing power is proportional to the clock frequency, doubling the clock frequency will roughly double the performance of the processor. However due to current consumption and other compatibility issues, it is not always possible to maximize the clock frequency. Other components for improving the performance include DMA (Direct Memory Access), MMU (Memory Management Unit), Java Accelerator, Coprocessors for symmetric and/or asymmetric cryptographic algorithms, and many others. Figure 20 illustrates a block diagram of high performance smart card microcontroller.

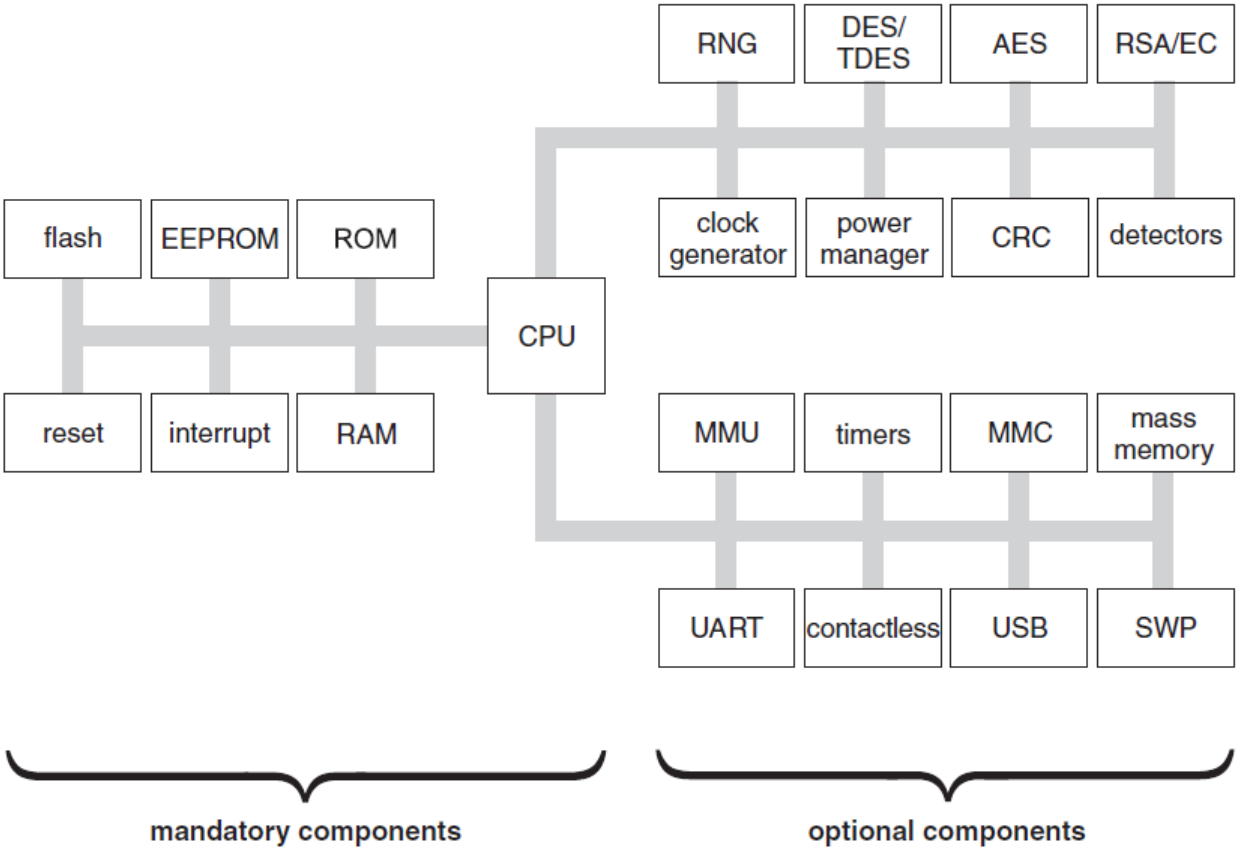

Figure 20. the usual functional units of a high-performance smart card microcontroller

3.3.3. Contactless Smart Cards

Contactless Smart cards refers to cards that do not require an electrical connection with the terminal to exchange data, power, and clock signal. All transmissions are done wirelessly over a short distance. Contactless smart cards must follow the smart cards standard - ID-1 format. The lack of electrical connection alongside the infeasibility of adding batteries to the card means the only way to transfer power to the card is via using the passive technique, in which the terminal provides the power signal through electromagnetic waves and the smart cards harness this power. This could be done in several ways, the most commonly used methods are radio waves or microwaves, optical transmission, capacitive coupling, and inductive coupling.

The main interests lies within the capacitive and inductive coupling since they are the most suitable for the flat shapes of the smart cards, also they are similar to the coupling techniques used in NFC devices. As seen in Figure 21.

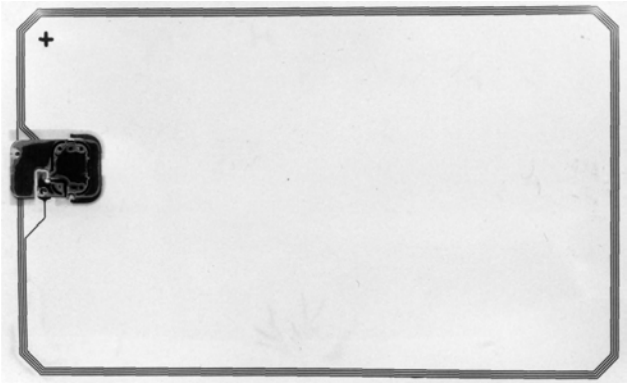

Figure 21. An inlay foil for a contactless smart card with inductive coupling using an etched coil

All cards that employ inductive coupling share the same principle. Coils enclose a large area of the cards are incorporated in the cards body, they serve as the antenna for receiving the power signal and also for sending and receiving data signals from and to the card. The coils are connected to one or more chip depending on the design of the smart card. (Rankl & Effing 2010: p285)

Capacitive coupling utilizes a different technique to achieve coupling. Conductive surfaces are incorporated in the cards body and in the terminal as well. These surfaces act as the two sides of a capacitor when the card is in close proximity to the terminal. The capacitance that can be obtained depends on the sizes of the coupling surfaces and the distance of the separation, but since the size of the card is limited by the ID-1 Format standards, considering the manufacturing costs, several picofarads can be obtained from a card manufactured at an acceptable level of cost and effort. This is not enough to power the microcontroller inside the smart card, hence this method is used only for data transmission, while the operating power is obtained by an inductive coupling. This mixed method is standardized in ISO/IEC 10536 for close-coupling cards, and as the name suggests, this method is limited to short coupling distances.

For collision avoidance two standards have been defined in ISO/IEC 14443-3, both based on time division multiple access (TDMA), they ensure that individual cards have

different timing behaviors, so they can be distinguished from one another at the terminal.

Due to the large number of applications in which contactless smart cards can be used, also the different optimization and customization each application requires. ISO and IEC began a standardization process for contactless smart cards in 1988.

Contactless smart cards can operate in range of one centimeter to one meter, this range has a significant effect on the transmission of power and data. Consequently, it is not possible to cover all contactless cards by one standard which offers technical solution to all of the requirements needed from the various applications. Presently, there are three different standards describing the contactless cards, each one is defined for a specific distance range, as mentioned in Table 5

Table 5. Completed ISO/IEC standards for contactless smart cards. Each standard consists of several parts

Standard	Type of contactless smart card	Range	Category
ISO/IEC 10536	Close coupling (CICC)	≈ 1 cm	Close Coupling Cards
ISO/IEC 14443	Proximity coupling (PICC)	≈ 10 cm	Remote Coupling Cards
ISO/IEC 15693	Vicinity coupling (VICC)	≈ 1 m	Remote Coupling Cards

Close integrated circuit cards (ISO/IEC 14443)

Close coupling utilizes capacitive coupling for data transmission and inductive coupling for powering the integrated circuit in the cards body. The ISO/IEC 10536 standard is designated for “Slot or surface operation”. Indicating the card must be inserted in a slot or laid on a marked area of the terminal, due to its short range (maximum of 1 cm).

The term “remote Coupling” encompasses smart cards that can transmit data over a range extending from few centimeters to approximately one meter from the terminal. Inserting the card or laying it on the marked surface on the terminal is not required, offering more flexibility and various applications. Examples of such applications are access control, vehicle identification, electronic tickets, local public transport, ski passes, airline tickets, electronic purses, and baggage identification. (Rankl & Effing 2010: p296). The diversity of these applications requires many different technical

implementations. The ISO/IEC 14443 and ISO/IEC 15633 standards address coupling ranges up to 10 cm and 1 m, respectively.

Proximity integrated circuit cards (ISO/IEC 14443)

The ISO/IEC 14443 standard, which is titled ‘Identification cards – Contactless integrated circuits cards – Proximity cards’, describes the properties and operating principles of contactless smart cards with a range of approximately 10 cm. The amount of power that can be transmitted over this range is sufficient to operate a microprocessor. A large number of contactless cards have also contacts in addition to coupling components, in order to make them compatible with an existing infrastructure for contact cards. This type of cards is called “dual-interface cards”. The ISO/IEC 14443 standard is formulated such that it is compatible with ISO/IEC 7816 (the standard for contact smart cards) at the application level, to ensure the formats for exchanging data between the card and the terminal are the same. Hence commands and data can be exchanged between the card and the reader using the contact or the contactless interface. (Rankl & Effing 2010: p297)

The maximum operational range is 10 cm might not be suitable for all applications, but it has other advantages which make these Proximity cards useful. The short range of operation provides a certain amount of protection against undesired access to the card. Large antennas in the immediate vicinity of a terminal are necessary for eavesdropping on data transmissions, this could hardly be installed in an unobtrusive manner. In addition to encryption and authentication, the immunity against attacks is very robust.

Several amendments were introduced to the standard in order to increase the data rate among other requirements. The clarity of the standard is reduced by the large number of amendments. Therefore, the responsible ISO/IEC group is working on a revised version of the standard.

Vicinity integrated circuit cards (ISO/IEC 15693)

ISO/IEC 15693, ‘Identification cards – Contactless integrated circuits cards –Vicinity cards’, describes the properties and operating modes of contactless smart cards with a range up to 1 meter. Complying with ISO/IEC 14443, the minimum activation field strength needed for vicinity cards is reduced, due to the restriction on the maximum allowable magnetic field strength. The reduced field strength leads to inefficient power to operate the microcontrollers in the cards. For that reason, only simple memory ICs with relatively simple security logic (a state machine) can be used. There is also a security issue associated with the larger working range. Thus it is possible for a terminal to establish a connection to a card without the desire or knowledge of the cardholder. Whether such undesired and unnoticed card accesses pose security risks depends on the application. (Rankl & Effing 2010: p344)

NEAR FIELD COMMUNICATION (NFC)

Proximity Coupling Cards and Near Field Communication (NFC) share the same standards (ISO/IEC 14443). Although NFC has some other standards on its own, the overlapping of these two technologies in several applications is the main reason to study them in parallel. Just as in Proximity coupling cards, the short range of operation in NFC devices prevents unintentional data transfer between equipment and terminal and is thus suitable for use in many applications with relatively simple security mechanisms. NFC Protocols has been covered in the previous titles.

The use of NFC devices on contactless terminals is possible, because contactless terminals are not dependent on the ID-1 Card Format. Instead NFC device must be located within range of the terminal. NFC devices, contactless cards and contactless terminals support the ISO/IEC 14443 standards. (Rankl & Effing 2010: p348)

One of the main target applications for NFC technology is contactless payment using mobile devices. To achieve that goal, security component in the NFC-capable mobile device is a prerequisite for mobile payment. Three options for integrating a security component in a mobile device are currently under discussion.

The first two options share the same concept of integrating a security component in the mobile device (Secure Element). The first option this element could be an already approved smart card microcontroller to host the payment application. The microcontroller is permanently integrated in the mobile device. To achieve the same security level as with a corresponding smart card. One problem with this solution is maintaining the interchangeability of the mobile equipment, since the application is permanently tied to the equipment. In the second option the secure element is an external secure memory card (such as a secure microSD card). In this case, the security chip is stored in the memory card next to the flash memory. The user can transfer the payment application to a different mobile device by changing the memory card.

In the third option, the secure element is not a part of the mobile device, the SIM or USIM card contains the payment application. The application can be stored directly in the SIM microcontroller or in a supplementary security chip in the SIM or USIM module. A supplementary security chip has the advantage that only this chip needs to be certified.

In any case, it is necessary to devise a solution for secure loading and personalization of the payment application in the security component. This process must be performed by a trustworthy entity that is certified for payment systems. (Rankl & Effing 2010: p351). Several examples are displayed in Figure 22.

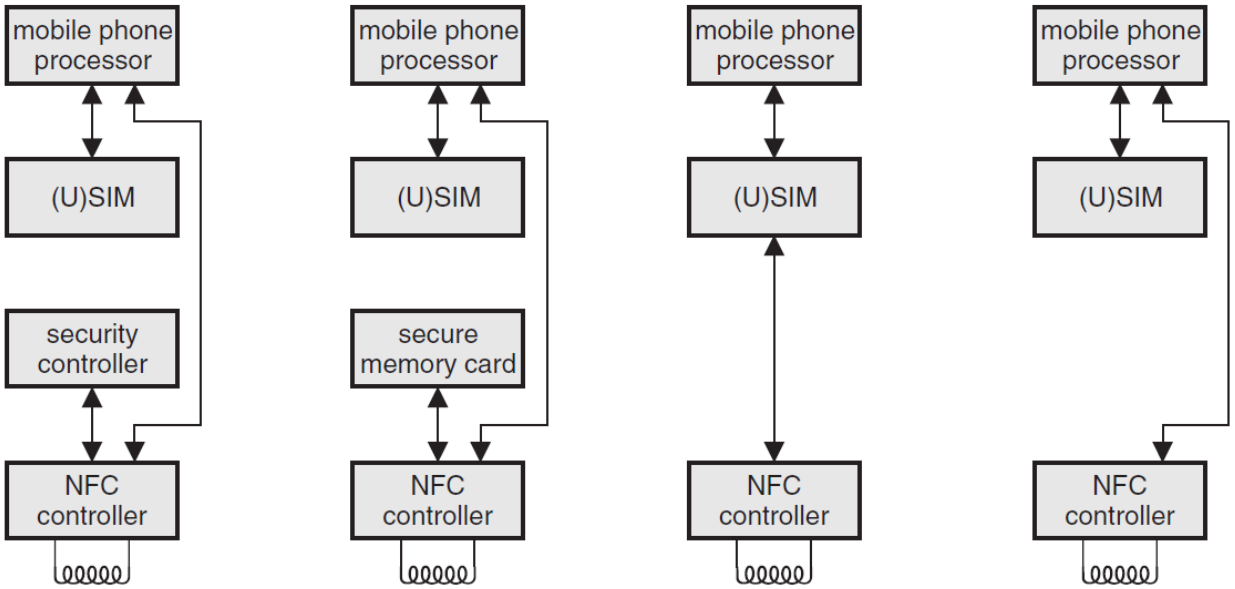

Figure 22. Several options for integrating a security component for NFC in a mobile device

The expanding spread of NFC and RFID systems in different applications necessitate the use of security measures to protect against attempted attacks. Modern encryption and authentication protocols employ suitable algorithms to prevent unauthorized access or use of the RFID systems. High-security RFID systems must have a defense against the following individual attacks:

• Skimming of a data carrier in order to clone and/or modify data.

• Placing a foreign data carrier within the interrogation zone of a reader with the intention of gaining unauthorized access to a building or receiving services without payment.

• Eavesdropping on radio communications and replaying the data, in order to imitate a genuine data carrier ‘replay and fraud’.

Consideration should be given to cryptographic functions when selecting a suitable RFID system. Applications that do not require a security function (e.g. industrial automation, tool recognition) would be made unnecessarily expensive by the incorporation of cryptographic procedures. On the other hand, in high-security applications (e.g. ticketing, payment systems) the omission of cryptographic procedures can be a very expensive oversight if manipulated transponders are used to gain access to services without authorization. (Finkenzeller 2010: p226)

Within the scope of this work, several algorithms will be analyzed from different point of view. In order to establish a basic understanding of the cryptographic abilities of RFID tags, NFC tags and smart cards. Tests will include brute force attack, encryption and decryption times, power consumed during encryption and decryption, and an overall performance analysis.

4. Practical Part

4.1. Encryption Algorithms

The wide variety of Cryptographic algorithms can be split into two major categories, symmetric and asymmetric. Symmetric algorithms use the same key for encryption and decryption, while asymmetric algorithms (which were first postulated in 1976 by Whitfield Diffie and Martin E. Hellman) use different keys for encryption and decryption.

The algorithms chosen to be tested are Caesar, DES/3DES, AES, and Blowfish.

4.1.1. Caesar

Also known as Shift cipher, Caesar cipher is one of the oldest and most known encryption techniques. In which each letter is “shifted” by a fixed number of letters, so a shift of 3 will make the letter “a” become “d”. The method is named after Julius Caesar, who used it in his private correspondence. In the original form the range of letter which can be shifted was limited to the number of letters in the alphabet. Nowadays although the Caesar cipher is considered the weakest form of encryption, in the computer world the shift range is wider than the original, at a range of (32-126), it includes letters, numbers, and special symbols such as “#” and “space”. the minimum effort needed to break the code is the main reason why the Caesar cipher is not used. Applying brute force attack on the ciphered text can resolve in finding the key. Even without the use of a computer, a brute force attack on a Caesar ciphered text is carried out by either shifting the ciphered text one letter at a time until the right key is found, or by using frequency analysis which will even shorten the time needed for breaking the code. In each language there are several letters which are used more than others, in the English language for example the letter “e” is the most reoccurring letter in the alphabet and by graphing the most occurring letter in a ciphered text to the most occurring letter in a certain language, and the code is

broken. Figure 23 display the frequency of letters in the English language. (http://en.wikipedia.org/wiki/Caesar_cipher)

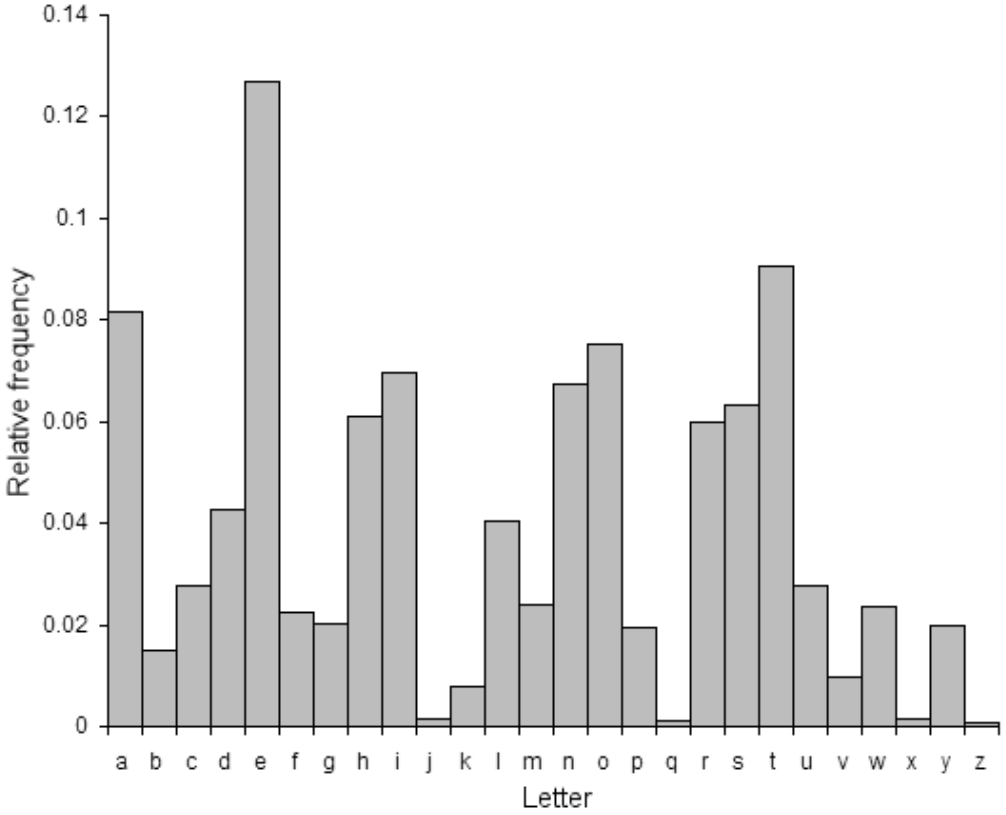

Figure 23. The distribution of letters in a typical sample of English language text.

4.1.2. DES/3DES

Developed by IBM in collaboration with the US National Bureau of Standards (NBS) and published in 1977 as the FIPS 46 standard. DES was fashioned in accordance with Kerckhoff's principle, which meant it could be published without impairing its security. DES is a symmetric block encryption algorithm that does not expand the ciphertext, which means that the plaintext and ciphertext blocks have the same size. The block size is 64 bits (8 bytes), which is also the key size, although only 56 of these bits are used as the actual key. Coding is done in 19 phases of substitution and permutation operations. The Figure 24 illustrate the general phases in the DES encryption. (http://dc242.4shared.com/doc/ZDomPSx_/preview.html)

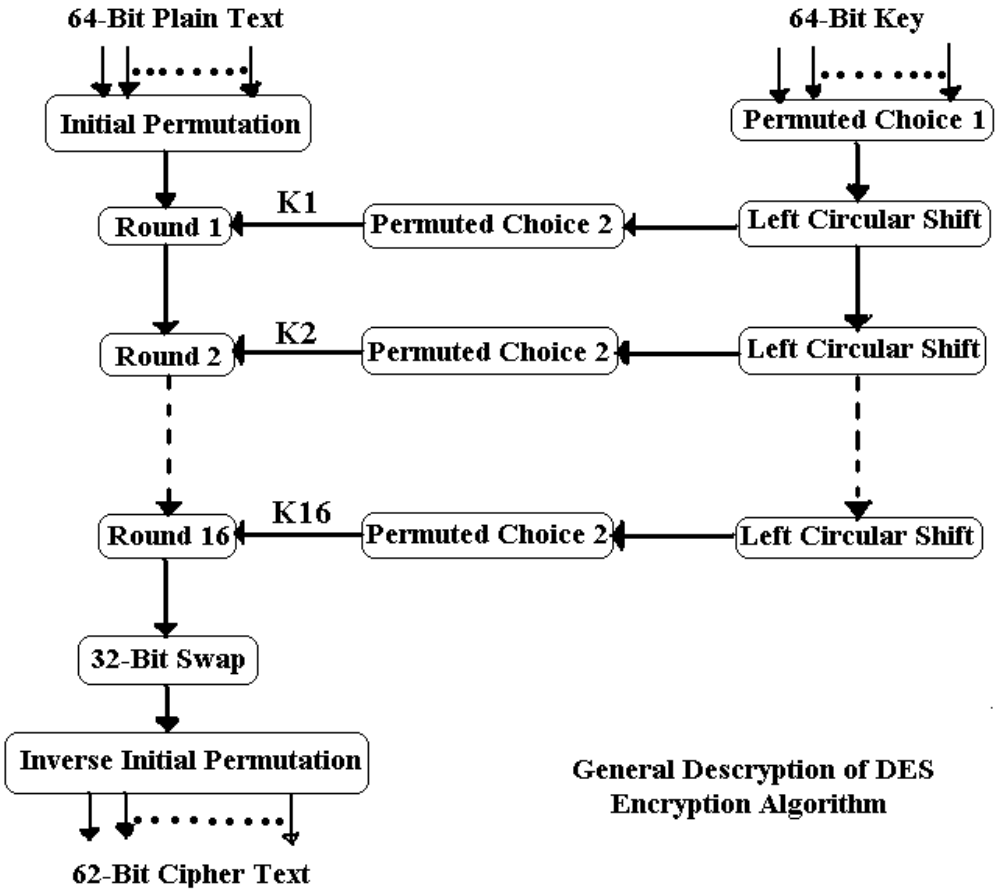

Figure 24. General description of DES encryption algorithm.

Before the encryption starts, the key is extended from 56 bits to 64 bits, and then rotating the key generates a new keys K_i for each phase (i = 1, . . . , 16).

The text is permutated in the first phase of the encryption, after that the rounds of permutations, begins, using the corresponding K_i generated from the originally extended key. After 16 round of permutation, the two halves of the block (two 32 bits blocks) are interchanged, and in the final phase, the first permutation is revered.

The security of DES can be increased by triple DES coding “3DES”. In 3DES, two keys K_1 and K_2 are needed. The way to change DES to 3DES is firstly the Plaintext block is coded by K1. Secondly, Result is decoded by K2. (Note as K2 is the wrong key, the result is not the original text but even more mixed.). And finally the previous result is coded again by K1. This is the result of 3DES. Decoding is done in reverse order using the same keys. (Penttonen 2011)

4.1.3. AES

AES is a symmetric block encryption algorithm with a block length of 128 bits (16 bytes) that can be used with three different key sizes, thus called AES-128, AES-192 or AES-256, depending on the key size (the number denote the key size). AES is suitable for hardware implementation, and can also be implemented in software running on low-performance 8-bit processors or high-performance 16-bit and 32-bit processors.

The size of the key space of AES with a 128-bit key is 2128 (≈ 3.4×1038), which is a factor of 4.7×1021 larger than the key space of DES with a 56-bit key. The larger key space of the AES grants higher security level against known attacks.

The basic structure of AES is substitution-permutation network. The cipher takes the plaintext block size of 128 bits. The key sizes can be 128, 192 or 256 bits. (Wikipedia AES 2012a.)

Figure 25 illustrates the general steps of encrypting plain text using AES with different key sizes.
(http://developer.amd.com/resources/documentation-articles/articles- whitepapers/bulk-encryption-on-gpus.)

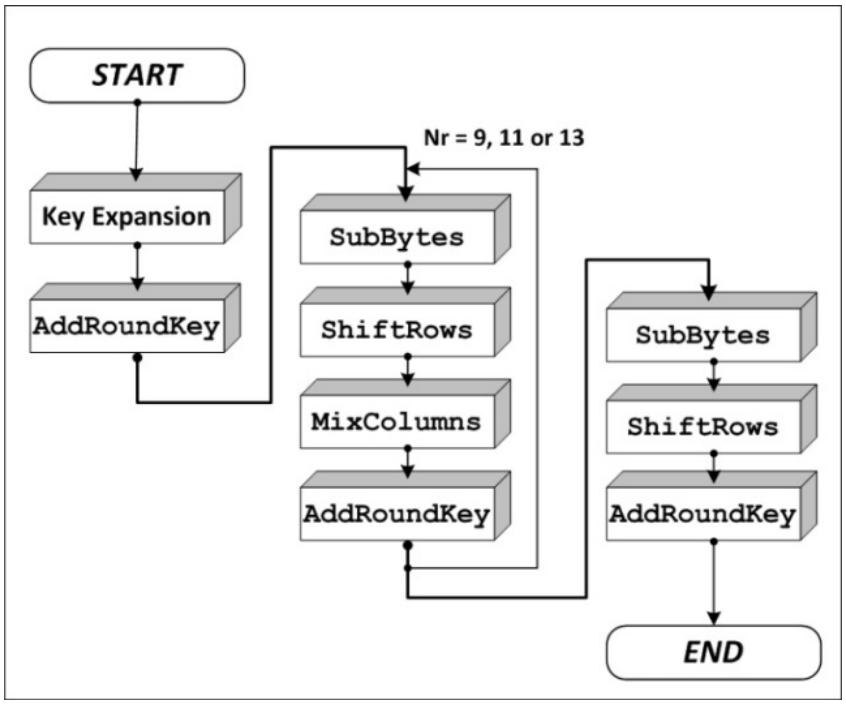

Figure 25. General description of DES encryption algorithm.

Key expansion means that the provided key as input is expanded into an array of forty-four 32-bit words. Four distinct words (128 bits) serve as a round key for each round.

There are 10-14 rounds in AES depending on the key size, each round has several steps:

• Substitute bytes: Uses a Substitution box to perform a byte-by-byte substitution of the block.

• ShiftRows: is the row forward shift process. The first row remains the same. For the second row, shift to left 1-byte circular. For the third row, shift to left 2-byte circular. Then the fourth row, shift to left 3-byte circular.

• MixColumns: is a forward mix column transformation. A substitution that makes use of arithmetic over Galois Field (2^8)

• AddRoundKey: A bitwise XOR of the current block with a portion of the expanded key.

The final round is different because it omits the “mix columns” step.

The round key is used only in the “AddRoundKey” step, which is why the cipher begins and ends with an “AddRoundKey” step. Any other step applied at the beginning or end, is reversible without knowledge of the key and so would add no security.

As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES. (Stalling 2010)

4.1.4. Blowfish

Blowfish is a substitute for the DES and IDEA encryption algorithm. It is a symmetrical block cipher (secret or private key), use that a variable key length from 32 to 448 bits. (The U.S. government prohibits the encryption output software to use the key which key-length is more than 40, unless special-purpose software).

Blowfish algorithm is an alternative encryption method, proposed in 1993 by Bruce Schneier. After the birth of the 32-bit processor, the speed of blowfish algorithm in the encryption beyond the DES attracted the attention of the people. Blowfish is a not registered patent, it can be used free.

There are some features of blowfish:

- Blowfish is fast, can be executed in 18 clock cycle in a 32-bit processor.
- Blowfish needs only 5 KB of memory to implement
- Blowfish is considered secure due to the key's adjustable length (32-448)
- Encryption consist 16+1 phases, each phase consists of $\oplus$, + and S-box operation
- Decryption is identical to encryption; keys are used in inverse order. (Penttonen 2009: p35.)

(Qian 2013)

Figure 26 and 27 shows the round function (Feistel function) of Blowfish encryption. (http://www.embedded.com/design/configurable-systems/4024599/Encrypting-data-with-the-Blowfish-algorithm)

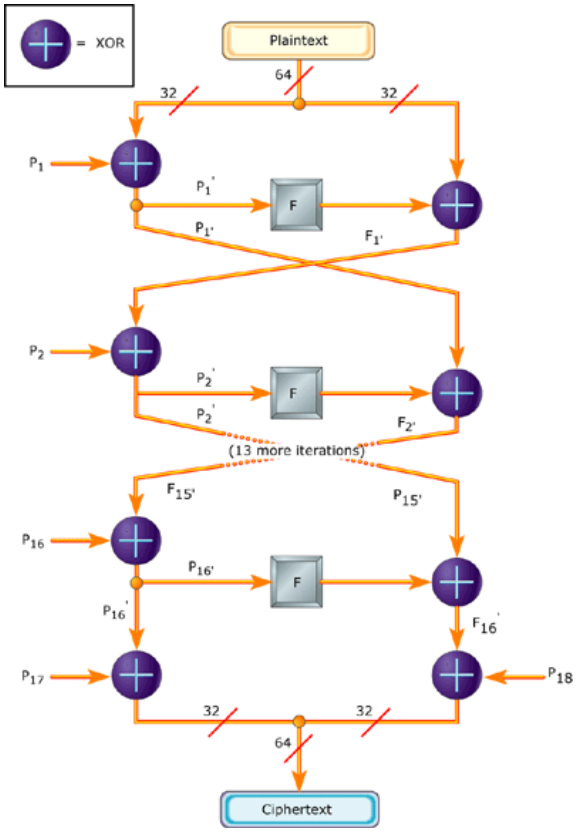

Figure 26. Blowfish Algorithm

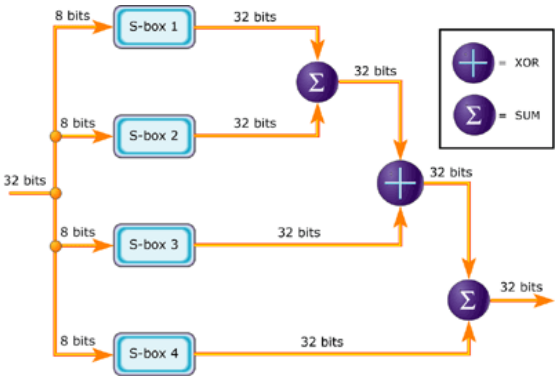

Figure 27. Blowfish Algorithm

4.1.5 Elliptic Curve Cryptography

Although elliptic curves cryptography will not be tested within the scope of this work, basic concepts will be illustrated for future references and work. Unlike the previous encryption algorithms, Elliptic Curves Cryptography (ECC) is an asymmetrical encryption algorithm.

Proposed in 1985 by Victor Miller and Neal Koblitz independently, ECC is considered more efficient than other asymmetrical encryption algorithms such as RSA, for the same level of encryption strength, ECC requires a smaller key size than RSA, which yields to faster encryption, lower power consumption and lower processing power.

Table 6 illustrates the keys sizes of ECC and RSA encryption algorithms, corresponding to similar encryption strength.

Table 6. Comparing ECC keys with RSA keys (Qian 2013)

ECC Key Length (bits)	RSA Key Length (bits)	Crack Time /MIPS (years)	ECC/RSA key length rate
106	512	10^4	5:1
160	1024	10^{11}	7:1
210	2048	10^{20}	10:1
600	21000	10^{78}	35:1

This cryptographic strength and the relatively small size of the keys are the reasons why ECC systems are used in the smart card environment. Table 7 shows different generation and verification time on different platforms.

Table 7. ECC algorithms on different Platforms (Rankl and Effing 2010)

Implementation	Generate a 160-bit signature	Verify a 160-bit signature
Smart card with 3.5-MHz clock and 8-bit processor	1 s	4s
Smart card with 3.5-MHz clock and numeric coprocessor	150 ms	450 ms
PC (Pentium III, 500 MHz)	10 ms	20 ms

Elliptic curves are continuous planar curves that satisfy the equation $y^2 = x^3 + ax + b$ in a finite three-dimensional space. No point on the curve is allowed to be a singularity, which for example means that $4a^2 + 27b^2 \neq 0$. The finite bodies GF(p), GF(2^n) and GF(p^n) are used in cryptography, where p is a prime number and n is a positive integer greater than 1.

In order to describe the ECC algorithm, Weierstrass equation is commonly used.

$$y^2 + axy + by = x^3 + cx^2 + dx + e \quad (1)$$

a, b, c, d are real numbers and x, y take the values of real numbers, for simplicity, the equation is reduced to

$$y^2 = x^3 + ax + b \quad (2)$$

To plot this equation, y is computed as

$$y = \sqrt{x^3 + ax + b} \quad (3)$$

For given values of and, the plot consists of positive and negative values of for each value of y. Thus, each curve is symmetric about y=0. This yields for any specific value of a and b, there is a set of points (E) which satisfy the equation number (X3), this set (E) also include a point called *O,* which is called the “zero point” or “point at infinity” which serves as the identity element of the group.

There are several implementations which use ECC as a foundation such as Elgamal ECC and Deffie-Hellman which is described in the following steps

A large integer Q is chosen, Q must a prime or an integer in the form of 2^m. The a and b parameters of the equation (X3) must be applied in order to generate a group of points $E_q(a,b)$. In the next step a base point G= (x_1 , y_1) in E_q (a,b)is selected, whose order is larger than value n. A key exchange between user Person A and B is described as follow

Person A Key Generation

Select private n_a where $n_a < n$

Calculate public P_a $P_a = n_a \times G$

Person B Key Generation

Select private n_b where $n_b < n$

Calculate public P_b $P_b = n_b \times G$

User A Calculate the secret key

$$K = n_a \times P_b \quad (4)$$

User B Calculate the secret key

$$K = n_b \times P_a \quad (5)$$

ECC Diffie-Hellman Key Exchange (Stallings 2011: 343).

ECC Encryption and Decryption

An Elliptic curve is chose over a certain Galois Field (e.g. GF(2^m)), the preparation assignments are:

- Select GF(p)
- Select elliptic curve (e)
- Select base point G(x,y)
- Applied algorithm for transforming plaintext into the points of elliptic curve, called encryption process

- Generating the private and public keys between sender A and receiver B. Select one private key n, calculate the public key $P = nG = n(x, y)$. For A, the private key is n_a, and the public key is $P_a = n_a G = n_a(x, y)$. For B, the private key is B n and the public key is $P_b = n_b G = n_b(x, y)$

Encryption: A sends encrypted message to B

- Choose random number k $1 \leq k \leq p - 1$
- Get the corresponding points (x_m , y_m) by encoding the plaintext;
- Calculate the cipher text $C_m = \{k(x, y), (x_m, y_m) + kP_b\}$, and the cipher text here turns into two points on the elliptic curve.

Decryption: B decrypts the received message from A:

- Calculation
  - $\big((x_m, y_m) + kP_m\big) - n_b(kG)$ (6)
  - $= \Big((x_m, y_m) + k\big(n_b(x, y)\big)\Big) - n_b\big(k(x, y)\big)$ (7)
  - $= (x_m, y_m) + kn_b(x, y) - n_b k(x, y)$ (8)
  - $= (x_m, y_m)$
- Get the corresponding plaintext by decoding the points (x_m,y_m).

The study and implementation of elliptic curve cryptography is now becoming a focus in public-key cryptosystems. Its relies on the difficulty to solve the discrete logarithm of the elliptic curve Abelian group.

They way RFID systems (tags, smart cards, NFC devices) utilize these encryption algorithms is by employing them in “Mutual Authentication” procedure. Mutual authentication means that both the reader and the target authenticate one another. The reader authenticates the target in order to protect the application from “manipulation” using falsified data. Likewise, the target must protect the data stored in it from skimming or overwriting by unauthorized readers.

Based on the application at which the RFID system is deployed, the mutual authentication can be either symmetrical or asymmetrical (symmetrical or asymmetrical encryption). Mutual symmetrical authentication is used in applications where the number or transponders (targets) are limited, and all the transponders and readers that form part of an application possess a single identical *cryptographical key K,* once the transponder is detected by the reader, the reader sends a (GET_CHALLANGE) command to the transponder. The transponder responds to the command, by generating a *random number* R_A and sends it to the reader in a (response → Challenge-response) procedure. The reader now generates a random number R_B. Using the common secret key *K* and a common key algorithm e_k, the reader calculates an encrypted data block (token 1), which contains both random numbers and additional control data, and sends this data block to the transponder.

$$Token\ 1 = e_k(|R_B||R_A||ID_A|Text1) \tag{9}$$

The transponder decrypts the received token, and checks if the random number contained in the plaintext $\acute{R_A}$ correspond to the original random number R_A. If they are equal, the transponder confirms that the two common keys are the same (the transponder's key and the reader's key).

Another random number R_{A2} is generated in the transponder and this is used to calculate an encrypted data block (token 2), which also contains R_B and control data. Token 2 is sent from the transponder to the reader.

$$Token\ 2 = e_k(|R_{A2}||R_B|Text2) \tag{10}$$

The reader decrypts the received token (token 2) and checks if the received random number within the token $\acute{R_B}$ is equal to the random number generated at the reader R_B. If the two figures correspond, then the reader is satisfied that the common key has been proven. Transponder and reader have thus ascertained that they belong to the same system and further communication between the two parties is thus is authenticated and valid. (Finkenzeller 2010)

The mutual authentication has several advantages:

- The secret key is never transmitted, only the numbers encrypted using the secret key are exchanged.
- The random numbers are always encrypted at the same time. To minimize the possibility of calculating the secret key from invers transformation using R_A to obtain Token 1.
- The authentication process is not limited to a specific encryption algorithm (mutual symmetrical authentication process must use symmetrical encryption algorithm).
- The use of two different random numbers from two different sources ensures safety against “replay attacks”, eliminating the possibility of recording the authentication sequence.

Mutual symmetrical authentication process is not suitable in applications where the number of transponders (targets) is vast, which will increase the probability of the secret key being discovered, because such transponders are accessible to an uncontrolled numbers.

Authentication using a derived key can provide a significant improvement on mutual symmetrical authentication, by securing each transponder with a different cryptological key. To achieve this, the serial number of each transponder is read out during its production. A key K_X is calculated (→ derived) using a cryptological algorithm and a master key K_M, and the transponder is thus initialized. Each transponder thus receives a key linked to its own ID number and the master key KM.

The mutual authentication begins by the reader requesting the ID number of the transponder. In a special security module in the reader - the Security Authentication Module (S.A.M) - the transponder’s specific key is calculated using the master key K_M, so that this can be used to initiate the authentication procedure. The S.A.M normally takes the form of a smart card with contacts incorporating a cryptoprocessor, which means that the stored master key can never be read. (Finkenzeller 2010)

4.2. Experimental Procedures

The aforementioned cryptographic algorithms will be tested in order to determine encryption and decryption times for different key lengths and different text sizes. As well as robustness against brute force attacks. Also the amount of power required for each operation.

4.2.1. System Description

The system consists of model B Raspberry Pi with Broadcom BCM2835 SoC, 700 MHz low power ARM1176JZE-F processor, and 512 MB SDRAM, running Raspbian operating system. SainSmart Mifare RC522 Card Read Antenna RF RFID Reader IC Card Proximity Module, 13.56MHz is connected to the system as the RFID reader. The other device is a computer with a 64-bit Intel Core i7-4700MQ CPU clocked at 2.4 GHz, and 8 Gb of Ram, the operating system is 64-bit windows 8.1. the computer is used to mainly test the immunity of the encryption algorithms against brute force attacks, as well as to give context to the results obtained from the Raspberry Pi.

The software implementation of the encryption algorithms is done using python script, a separate program is written for each different encryption algorithms. The source codes can be found in the appendix. Figure 28 illustrates the general flowchart of the program.

The chosen algorithms are Caesar, DES, 3DES, AES, and Blowfish. Caesar is chosen for the sole purpose of illustrating the basic concept of encryption and decryption. DES, 3DES and AES are used currently in different applications where RFID systems are implemented, note that DES was used before it was replaced by 3DES and AES. And Blowfish is chosen for the flexible range of key sizes that can be used.

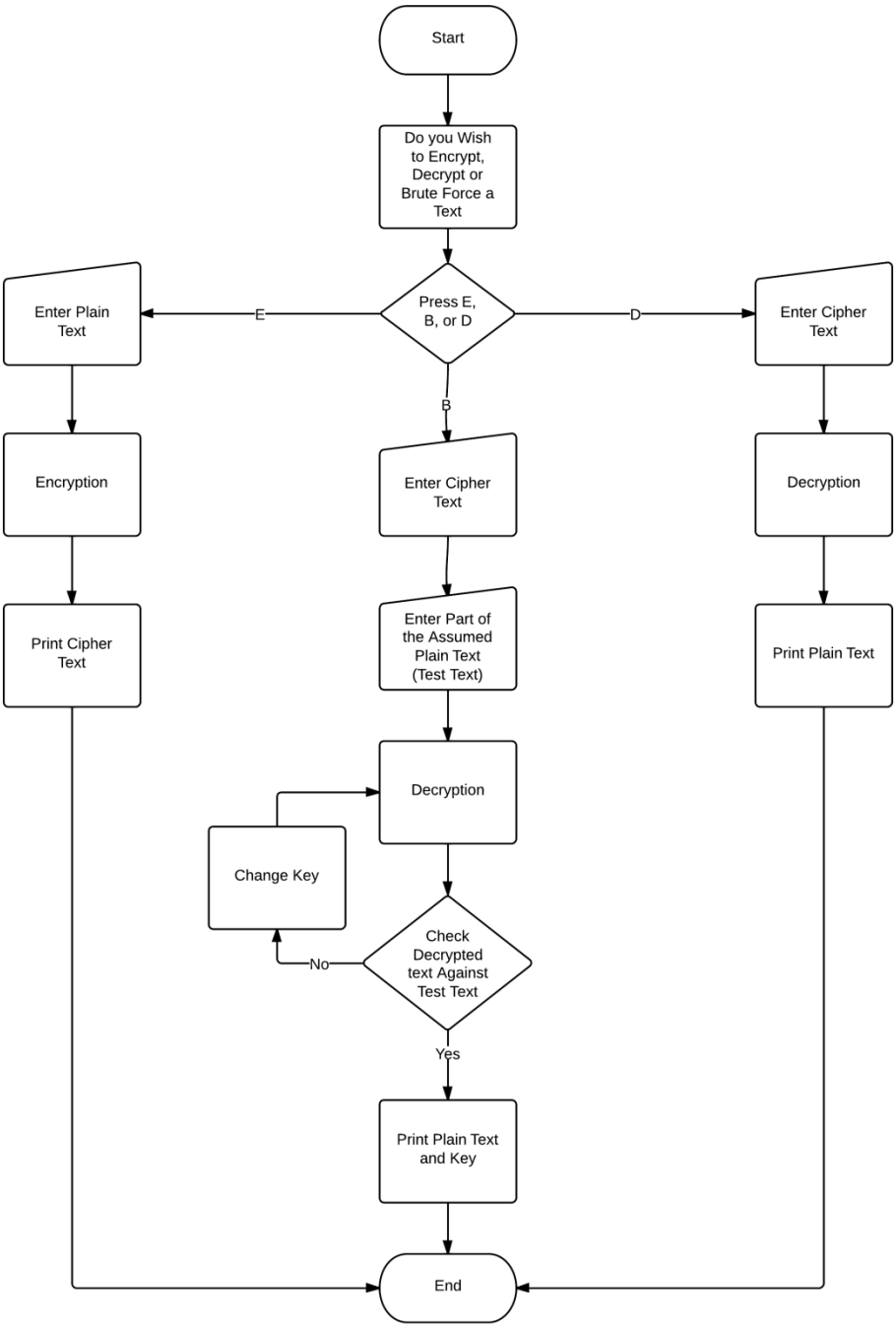

Figure 28. general flow chart of the used programs

Each algorithm will be run multiple times, with different key lengths and different text sizes. For each combination the test will be done five times, an average will be used to determine the final value. The same texts will be used and all tests are done under similar conditions.

The testing experiments are divided into two phases; phase one which is done using the windows machine. The results obtained are used as a reference to the one obtained from the Raspberry Pi. These tests are designed to take time measurement only, using the timer function provided by the Python libraries.

Phase two is redoing the same tests on the Raspberry Pi, the only difference is instead of entering the plain/cipher text manually, it will be read/written directly from/to the tags, the rest of the process is identical to the ones done on the windows machine.

Phase one tests are

1- Caesar 5 shifts
2- Caesar 10 shifts
3- Caesar 15 shits
4- Des 8 bytes key length
5- 3DES 16 bytes key length
6- 3DES 24 byte Key length
7- AES 16 byte key length
8- AES 24 byte key length
9- AES 32 byte key length
10- Blowfish 8 byte key length
11- Blowfish 16 byte key length
12- Blowfish 24 byte key length
13- Blowfish 32 byte key length

These tests are done over two different text sizes (16 bytes and 1024 bytes) and they include encryption time, decryption time, and brute force attack.

The text samples are

16 bytes : hello world!!

1024 bytes: The Snapdragon 810 replaces the 805 as the top of the line chipset. It features four Cortex-A57 and four Cortex-A53 processor cores. Those are the Cortex-A15 and A7 replacements respectively, but the A57 should offer a 25-55% increase in performance at the cost of just 20% increase in power consumption. And the power

consumption will probably actually be even as the 20% difference will be offset by the use of a 20nm manufacturing process.

The way the two new chipsets work is that the CPU cores are divided into two groups. All cores can work at the same time, but cores in a group must use the same frequency. This differs from the Krait designs where the clock speed of each core can be set individually.

In terms of GPU, the Snapdragon 810 has a brand new Adreno 430, which is advertised as 30% faster than the Adreno 420, which in turn is 40% faster than the Adreno 330 found in current Snapdragon 800/801 chipsets.The end result is something like an 80% performance increase over the current generation. filler

(http://www.gsmarena.com/qualcomm_unveils_snapdragon_810_and_808_64bit_chipsets-news-8241.php)

The results will be divided according to their respective algorithms in the results chapter.

Phase two is oriented towards the Raspberry Pi, since the Raspberry Pi platform share similar CPU architecture with modern mobile phones (ARM CPU Architecture), it can be used to represent a mobile device, the encryption and decryption tests will be conducted on the Raspberry Pi in order to obtain time measurements and power consumption measurements.

The key space used in the encryption algorithms is limited to decimal numbers, also the all brute force attacks on all algorithms will succeed in deciphering the encrypted text after one billion attempts. The reasons for this design choice are to compare different algorithms with different key lengths equally, since a larger key length by one byte multiplies the key space by ten times. Also, to check if different key lengths for the same algorithm, has any effect on encryption and decryption times.

5. Results

5.1. Phase one results

Phase one results are results obtained from running the algorithms on the computer machine.

5.1.1. Caesar

The cipher texts for the 16 bytes plain text are

5 shifts: mjqqt%|twqi&&

10 shifts: rovvy*"y|vn++

15 shifts: wt{{~/'~"{s00

Table 8 displays encryption, decryption and brute force attack results. While Figures 29, 30, 31 show these times in relationship to each other's.

Table 8. Encryption, decryption and brute force time for different shift on 16 bytes text

16 Bytes	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg**	**StdDev**
Encrypt	5	2.3093E-5	1.3257E-5	3.5495E-5	1.2402E-5	1.2402E-5	1.9330E-5	1.0104E-5
	10	1.5396E-5	1.3685E-5	1.3685E-5	1.2402E-5	1.2402E-5	1.3514E-5	1.2321E-6
	15	1.3685E-5	3.1646E-5	1.3685E-5	1.5396E-5	2.3093E-5	1.9501E-5	7.8250E-6
Decrypt	5	3.0791E-5	2.7370E-5	4.2338E-5	1.2830E-5	1.2830E-5	2.5232E-5	1.2607E-5
	10	2.5659E-5	3.6778E-5	1.7534E-5	3.1646E-5	3.7206E-5	2.9765E-5	8.2848E-6
	15	4.0199E-5	1.3685E-5	1.3257E-5	1.3257E-5	1.4968E-5	1.9073E-5	1.1831E-5
Brute Force Attack	5	6.7710E-3	9.3836E-3	1.0559E-2	1.1595E-2	1.4095E-2	1.0481E-2	2.7052E-3
	10	2.4662E-2	2.2888E-2	2.2311E-2	2.2778E-2	2.3723E-2	2.3272E-2	9.2901E-4
	15	2.2498E-2	2.2599E-2	2.2942E-2	2.3755E-2	2.2604E-2	2.2880E-2	5.1726E-4

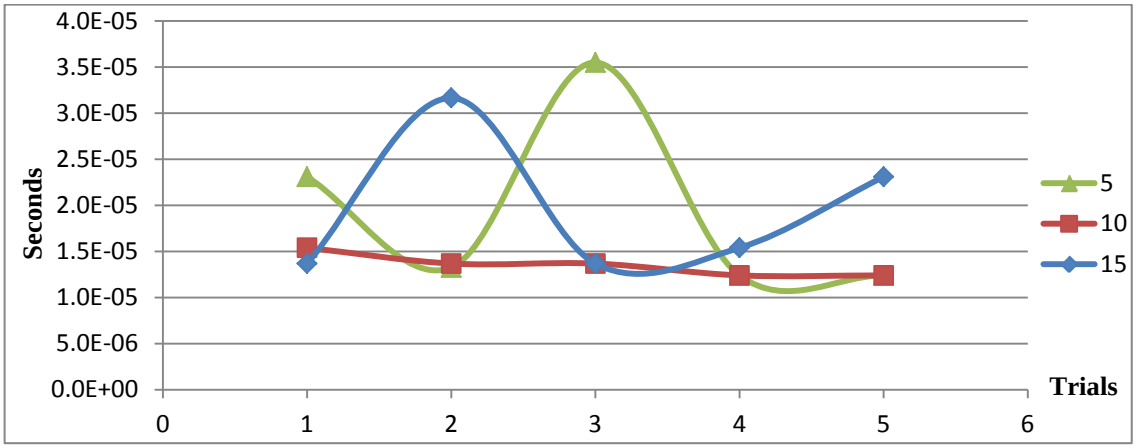

Figure 29. Encryption time over 5 trials.

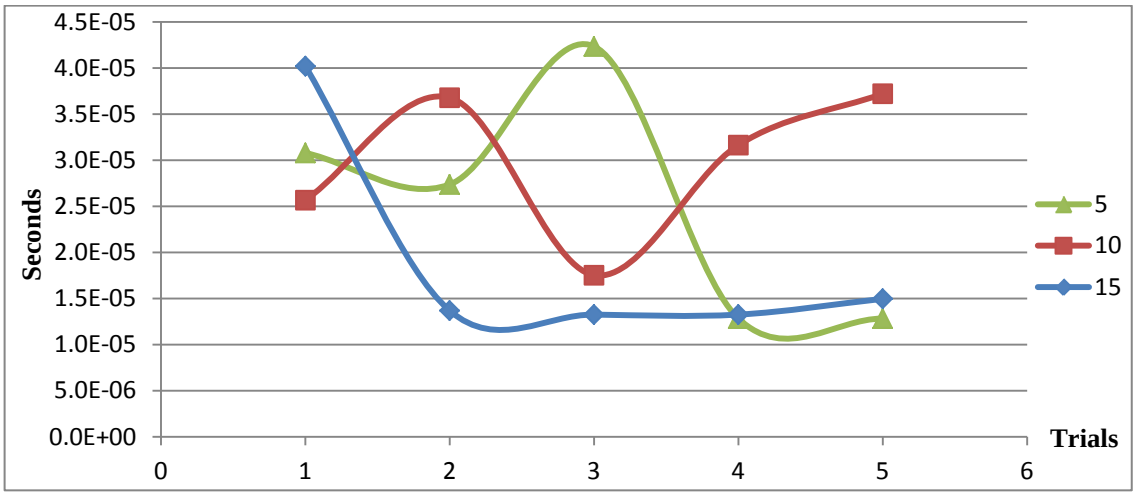

Figure 30. Decryption time over 5 trials.

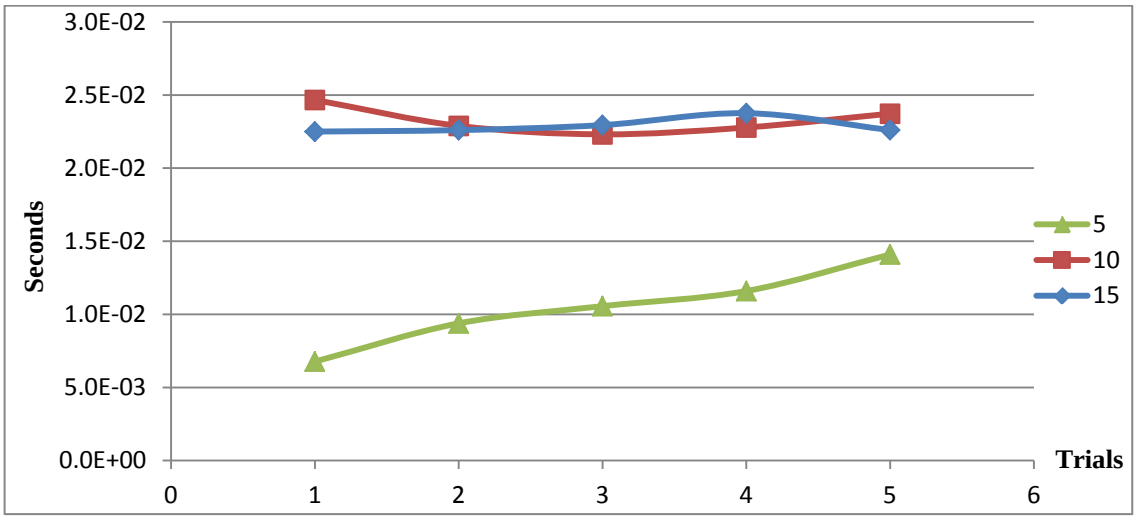

Figure 31. Brute force attack time over 5 trials.

The cipher texts for the 1024 bytes plain text are

5 shifts:

Ymj%Xsfuiwflts%=65%wjuqfhjx%ymj%=5:%fx%ymj%ytu%tk%ymj%qnsj%hmnuxjy
3%Ny%kjfyzwjx%ktzw%Htwyj}2F:<%fsi%ktzw%Htwyj}2F:8%uwthjxxtw%htwjx3%
Ymtxj%fwj%ymj%Htwyj}2F6:%fsi%F<%wjuqfhjrjsyx%wjxujhyn{jq~1%gzy%ymj%
F:<%xmtzqi%tkkjw%f%7:2::*%nshwjfxj%ns%ujwktwrfshj%fy%ymj%htxy%tk%ozxy
%75*%nshwjfxj%ns%ut|jw%htsxzruynts3%Fsi%ymj%ut|jw%htsxzruynts%|nqq%uwtg
fgq~%fhyzfqq~%gj%j{js%fx%ymj%75*%inkkjwjshj%|nqq%gj%tkkxjy%g~%ymj%zx
j%tk%f%75sr%rfszkfhyzwnsl%uwthjxx3

10 shifts:

^ro*]xkzn|kqyx*B;:*|ozvkmo}*~ro*B:?*k}*~ro*~yz*yp*~ro*vsxo*mrsz}o~8*S~*pok
~ |o}*py |*My|~o#7K?A*kxn*py
|*My|~o#7K?=*z|ymo}}y|*my|o}8*^ry}o*k|o*~ro*My|~o#7K;?*kxn*KA*|ozvkmowo
x~}*|o}zom~s!ov$6*l ~*~ro*K?A*}ry
vn*yppo|*k*<?7??/*sxm|ok}o*sx*zo|py|wkxmo*k~*~ro*my}~*yp*t
}~*<:/*sxm|ok}o*sx*zy"o|*myx} wz~syx8*Kxn*~ro*zy"o|*myx}
wz~syx*"svv*z|ylklv$*km~
kvv$*lo*o!ox*k}*~ro*<:/*nsppo|oxmo*"svv*lo*ypp}o~*l$*~ro* }o*yp*k*<:xw*wkx
pkm~ |sxq*z|ymo}}8

15 shifts:

cwt/b}p s"pv~}/G@?/"t {prt#/$wt/G?D/p#/$wt/$~ /~u/$wt/{x}t/rwx
#t$=/X$/utp$%"t#/u~%"/R~"$t(<PDF/p}s/u~%"/R~"$t(<PDB/
"~rt##~"/r~"t#=/cw~#t/p"t/$wt/R~"$t(<P@D/p}s/PF/"t {prt|t}$#/"t#
tr$x&t{);/q%$/$wt/PDF/#w~%{s/~uut"/p/AD<DD4/x}r"tp#t/x}/
t"u~"|p}rt/p$/$wt/r~#$/~u/y%#$/A?4/x}r"tp#t/x}/ ~'t"/r~}#%| $x~}=/P}s/$wt/
~'t"/r~}#%| $x~}/'x{{/
"~qpq{)/pr$%p{{)/qt/t&t}/p#/$wt/A?4/sxuut"t}rt/'x{{/qt/~uu#t$/q)/$wt/%#t/~u/p/A?}|/|
p}%upr$%"x}v/ "~rt##=

Table 9 displays encryption, decryption and brute force attack results. While Figures 32, 33, 34 show these times in relationship to each other's.

Table 9. Encryption, decryption and brute force time for different shift on 1024 bytes text

1024 Bytes	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg**	**StdDev**
Encrypt	5	2.4676E-4	2.3564E-4	2.7284E-4	2.4462E-4	1.9244E-4	2.3846E-4	2.9216E-5
	10	2.2965E-4	2.1939E-4	2.6472E-4	3.0834E-4	1.9287E-4	2.4299E-4	4.4692E-5
	15	2.3264E-4	2.4034E-4	2.4248E-4	2.4633E-4	2.5103E-4	2.4256E-4	6.8784E-6
Decrypt	5	2.3222E-4	2.7199E-4	2.4889E-4	2.2837E-4	1.9073E-4	2.3444E-4	2.9878E-5
	10	2.5060E-4	2.7755E-4	1.9458E-4	2.7755E-4	1.9031E-4	2.3812E-4	4.3147E-5
	15	2.3222E-4	2.3778E-4	2.4291E-4	2.2751E-4	2.3179E-4	2.3444E-4	5.9749E-6
Brute Force Attack	5	1.9052E-2	1.9383E-2	2.0368E-2	1.9394E-2	1.9366E-2	1.9513E-2	4.9916E-4
	10	3.2252E-2	3.0454E-2	2.8768E-2	2.9402E-2	2.9597E-2	3.0095E-2	1.3484E-3
	15	1.9719E-2	1.9395E-2	1.9985E-2	2.0172E-2	2.4259E-2	2.0706E-2	2.0078E-3

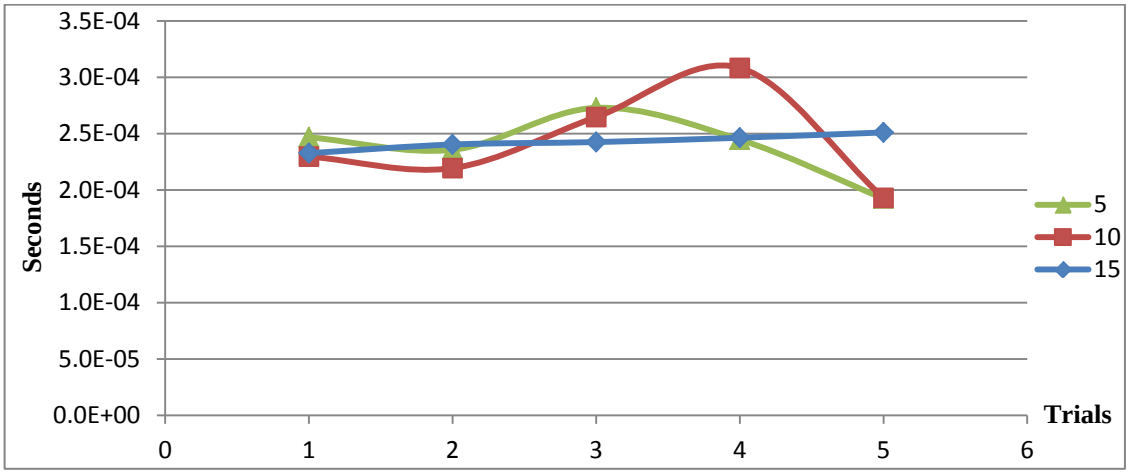

Figure 32. Encryption time over 5 trials.

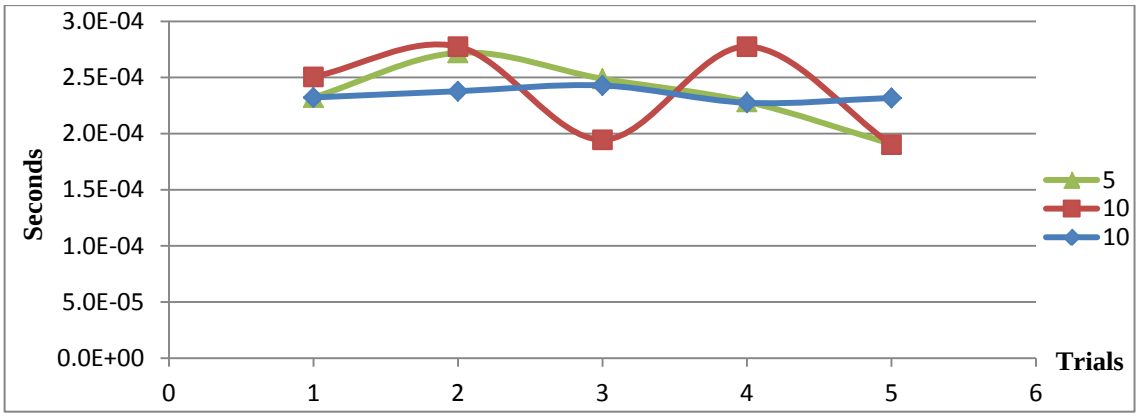

Figure 33. Decryption time over 5 trials.

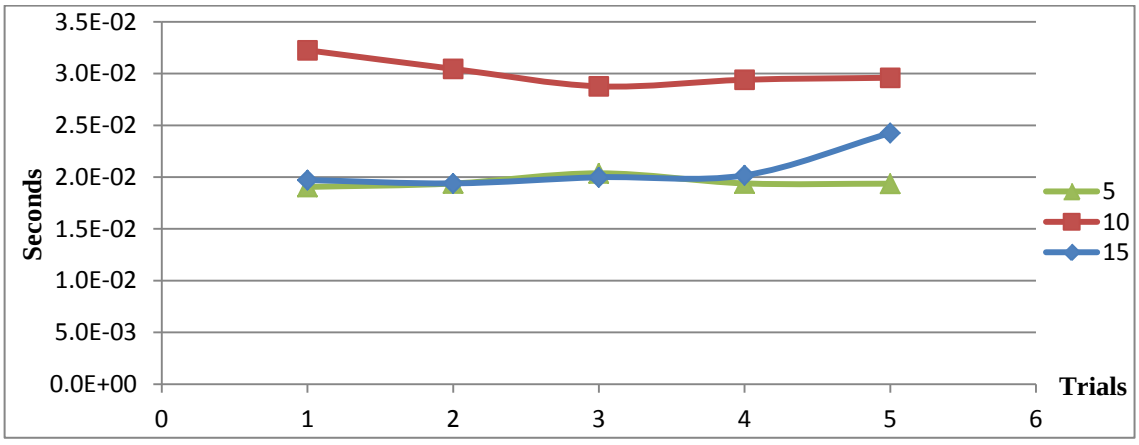

Figure 34. Brute force attack time over 5 trials.

Figures 35, 36, summarize the encryption and decryption times, while Figure 37 shows the brute force attack times

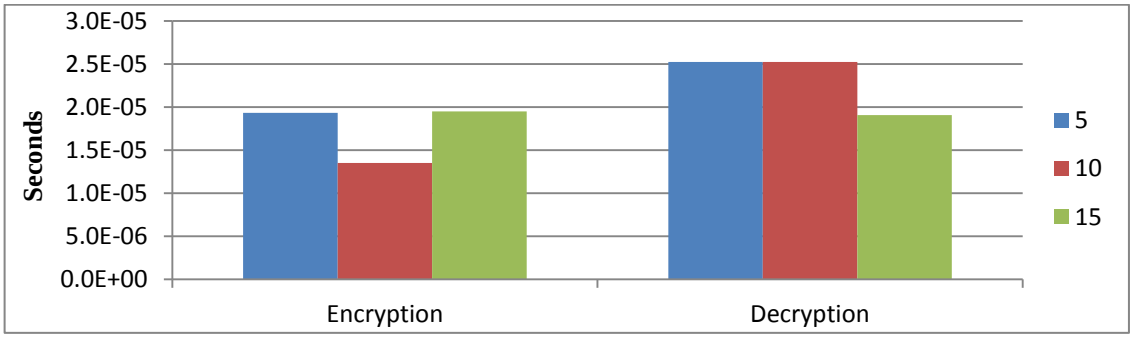

Figure 35. Encryption and decryption average time for 5, 10 and 15 Caesar shifts, text size 16 bytes

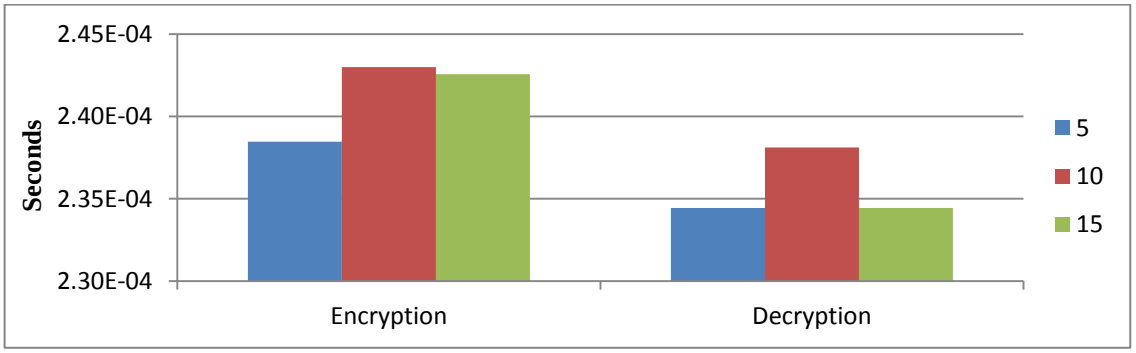

Figure 36. Encryption and decryption average time for 5, 10 and 15 Caesar shifts, text size 1024 bytes

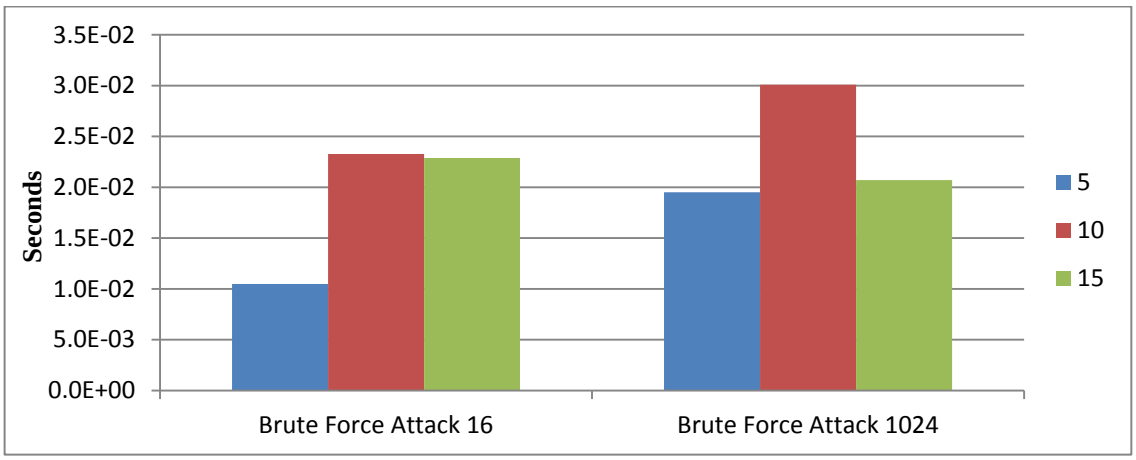

Figure 37. Brute Force Attack average times for 5, 10 and 15 Caesar shifts, text size 16 and 1024 bytes.

5.1.2. DES

DES has a fixed Key length of 8 bytes.

Used Key: 999999999

The cipher text for the 16 bytes plain text is

HNWuF7O23BJTtYvDiF2Jgg==

Tables 10 and 11 displays encryption, decryption and brute force attack results. While Figures 38 and 39 show these times in relationship to each other's.

Table 10. Encryption, decryption and brute force time for DES on 16 bytes text

16 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	StdDev
Encrypt	8B	1.2274E-4	5.0035E-5	5.2601E-5	5.1318E-5	5.1318E-5	6.5602E-5	3.1952E-5
Decrypt	8B	4.9608E-5	4.9180E-5	5.0891E-5	5.0035E-5	5.1318E-5	5.0206E-5	8.8680E-7
Brute Force Attack	8B	1.0559E+3	1.0190E+3	1.0347E+3	1.0182E+3	1.0201E+3	1.0296E+3	1.6206E+1

The cipher text for the 1024 bytes plain text is

82NYhRrd7EYnjHLoHYfx9J5HDf9BO/0/9ygzB/KKWN/HGaIkQeedKanDQSkulkTlR
O8tbeQw0OQmcJuAMgz08TLikX+Ep1x7BK0Ut3K7L4KQvpGD1UmkvTwuMdgeG
CMzaYuw4RYuJW020bVObj9jfQz3ShC/VjmytXvU71h63jec9ASIhjYb5Z9XzdFnvw
VihZYhT+HrAo4LhEUJxbVmLEnvRPaxnyNzXUMQyfthdKBCCZVkouSwc4hk9dIf7
DeBNLhZYXcF/MP++h2hp2tfRM6zyIt+8W5DnMN8UNcMLx2V9Q5yMrUon1gfQk
DuA0vR+jopUPsajAdlHd6P9spG707+uUePCQMk4DvccPuRs0WV9Q5yMrUon7jqXS
pZ2c7sJ0OkbC3lpz87dmwu1oHtSVnnu84YIEPR90rOTN05UteMMv2n0sHVykNyz2c
au2aw0ABPLyNs/WiyIfLx6Hr/WG7wB+TIgw1obldfgdRV/QZ45Ax7pFaZLQx3qkmd
ctnHesNrypXfX4TC9HQnL4NUfQ1bQgXi+lejpMeavcQQkMjt9RuB9yBV/8xrakaL3N
AXShrZOymPiUb22pt2EbfvBw==

Table 11. Encryption, decryption and brute force time for DES on 1024 bytes text.

1024 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg**	**StdDev**
Encrypt	8B	1.7106E-4	1.7491E-4	1.5652E-4	1.5652E-4	6.6286E-5	1.4506E-4	4.4819E-5
Decrypt	8B	1.4968E-4	1.5823E-4	6.3293E-5	1.3300E-4	1.3172E-4	1.2718E-4	3.7438E-5
Brute Force Attack	8B	1.7489E+3	1.7025E+3	1.7036E+3	1.6935E+3	1.7017E+3	1.7101E+3	2.2098E+1

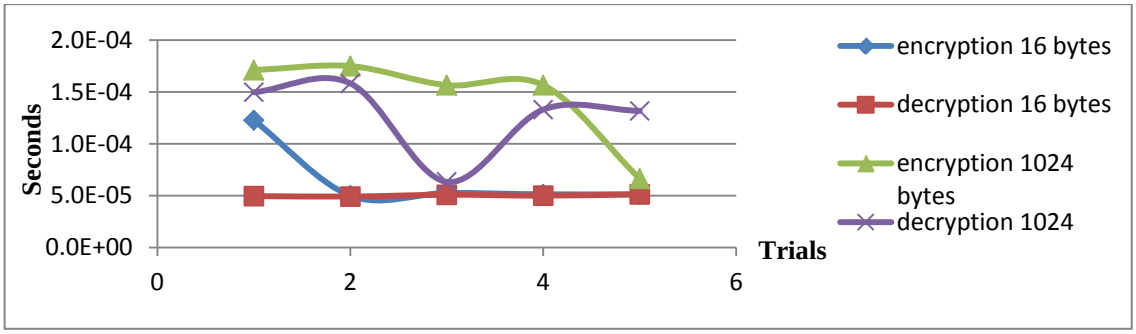

Figure 38. Encryption and decryption trials for DES, text size 16 and 1024 bytes.

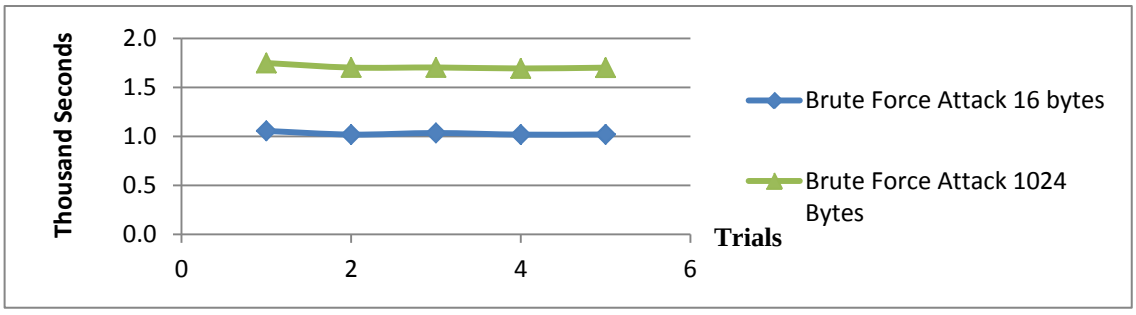

Figure 39. Brute Force Attack average times for DES, text size 16 and 1024 bytes.

Figures 40, summarizes the encryption and decryption times, while Figure 41 shows the brute force attack times.

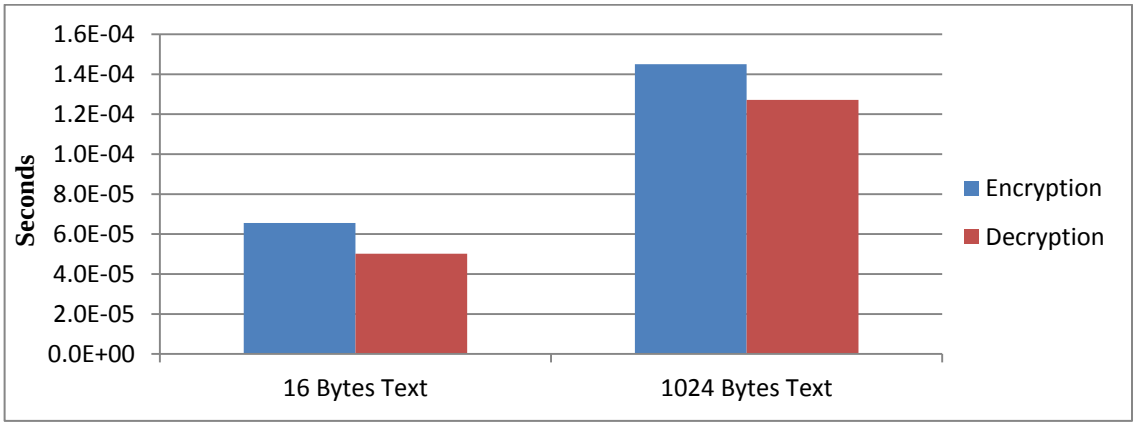

Figure 40. Encryption and decryption average time for DES, text size 16 and 1024 bytes.

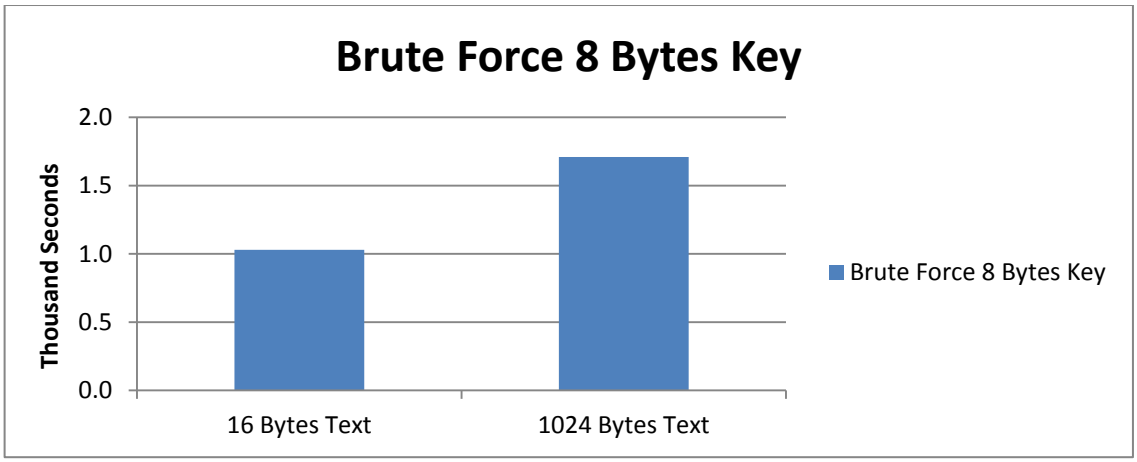

Figure 41. Brute Force attack average time for DES, text size 16 and 1024 bytes.

4.1.3. 3DES

Two keys were used 16 bytes (0000000099999999) and 24 bytes (000000000000000099999999)

The cipher texts for the 16 bytes plain text is

16 bytes key:

Bhtbz0KjVIne2AxRxFMO3A==

24 bytes key:

HNWuF7O23BJTtYvDiF2Jgg==

Tables 12 and 13 displays encryption, decryption and brute force attack results. While Figures 42 to 47 show these times in relationship to each other's.

Table 12. Encryption, decryption and brute force time for 3DES on 16 bytes text with different keys

16 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg**	**StdDev**
Encrypt	16B	6.4148E-5	6.4148E-5	1.0135E-4	6.4576E-5	8.6386E-5	7.6122E-5	1.7045E-5
	24B	6.6286E-5	1.2872E-4	6.3720E-5	6.5431E-5	6.2437E-5	7.7320E-5	2.8774E-5
Decrypt	16B	6.1582E-5	1.3813E-4	1.3813E-4	6.4148E-5	6.2865E-5	9.2972E-5	4.1235E-5
	24B	6.3293E-5	1.1333E-4	8.5103E-5	6.5003E-5	6.1582E-5	7.7662E-5	2.2095E-5
Brute Force Attack	16B	1.7335E+3	1.7348E+3	1.7464E+3	1.7313E+3	1.6750E+3	1.7242E+3	2.8131E+1
	24B	1.7388E+3	1.7811E+3	1.7294E+3	1.7420E+3	1.7333E+3	1.7449E+3	2.0802E+1

The cipher text for the 1024 bytes plain text is

16 bytes key:

Qi0Vrm9TY8lu8/rTr545PoPy0olth+beJ9245U5R2SNrGJvWUIwlj5fgijdWUiRLtTh1h
2hyxCK+La+96bRshAmnR0GSzUs0aH5f/e6J1oOZ+xW3Hcf0ebjg5BiD84teDa7HsVx
xPvCohUEWZ1MFRkn50qR5byh2tTHucYQeONT7QUMKZ/MqXDjJP6vpsLCe2bB4
oJtCiDD6VVlOEz/1YDPsRRg2LZHafzeYI4YCEUudLTomFxWwXDnKDsZLFX5hqx
RKJBcJGw+M2U2Kf27kBbIwzV7RDvN3uLApVTleO2j96hxVtjGZlNkpxeoRX/7Rw
u1MzIZc60Bv4oDXjYYuvvAVoWw4mqWumdzRUQ9y+3T96hxVtjGZlMkWzTCv3z
JYH+NF9xZUMUdR1uygrPuaZe8GgtL06dJsZuXb80KavKBTTDJyKbawln1NUFGN
YHkLcgrJHwaYdw/8OJsreSWYl2VdRFdZOHGEK/7tKr65n+6hrjl1XkA+PhJHSf62m
bbz7G2wfIMDygQv4hrofGnw7+Ys4cPFfeF6K1QqGjW2QHfphGkZc5wnJtcT8556yf
MCG3b0h83FbZowQ0NTce3Hkw==

24 bytes key:

82NYhRrd7EYnjHLoHYfx9J5HDf9BO/0/9ygzB/KKWN/HGaIkQeedKanDQSkulkTlR
O8tbeQw0OQmcJuAMgz08TLikX+Ep1x7BK0Ut3K7L4KQvpGD1UmkvTwuMdgeG
CMzaYuw4RYuJW020bVObj9jfQz3ShC/VjmytXvU71h63jec9ASIhjYb5Z9XzdFnvw
VihZYhT+HrAo4LhEUJxbVmLEnvRPaxnyNzXUMQyfthdKBCCZVkouSwc4hk9dIf7
DeBNLhZYXcF/MP++h2hp2tfRM6zyIt+8W5DnMN8UNcMLx2V9Q5yMrUon1gfQk
DuA0vR+jopUPsajAdlHd6P9spG707+uUePCQMk4DvccPuRs0WV9Q5yMrUon7jqXS
pZ2c7sJ0OkbC3lpz87dmwu1oHtSVnnu84YIEPR90rOTN05UteMMv2n0sHVykNyz2c
au2aw0ABPLyNs/WiyIfLx6Hr/WG7wB+TIgw1obldfgdRV/QZ45Ax7pFaZLQx3qkmd
ctnHesNrypXfX4TC9HQnL4NUfQ1bQgXi+lejpMeavcQQkMjt9RuB9yBV/8xrakaL3N
AXShrZOymPiUb22pt2EbfvBw==

Table 13. Encryption, decryption and brute force time for 3DES on 1024 bytes text with different keys

1024 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	**Avg**	**StdDev**
Encrypt	16B	1.9715E-4	1.8261E-4	1.3086E-4	2.0356E-04	9.1090E-5	1.6105E-4	4.8427E-5
	24B	2.3778E-4	2.2024E-4	1.9159E-4	2.0827E-04	1.2915E-4	1.9740E-4	4.1718E-5
Decrypt	16B	1.7662E-4	1.9587E-4	1.9672E-4	9.0235E-05	1.9244E-4	1.7038E-4	4.5531E-5
	24B	1.7149E-4	1.7577E-4	1.1205E-4	9.2373E-05	9.1090E-5	1.2855E-4	4.2004E-5
Brute Force Attack	16B	3.5570E+3	3.4344E+3	3.3370E+3	3.4656E+03	3.5697E+3	3.4727E+3	9.5458E+1
	24B	3.4481E+3	3.5587E+3	3.4563E+3	3.3912E+03	3.3853E+3	3.4479E+3	6.9799E+1

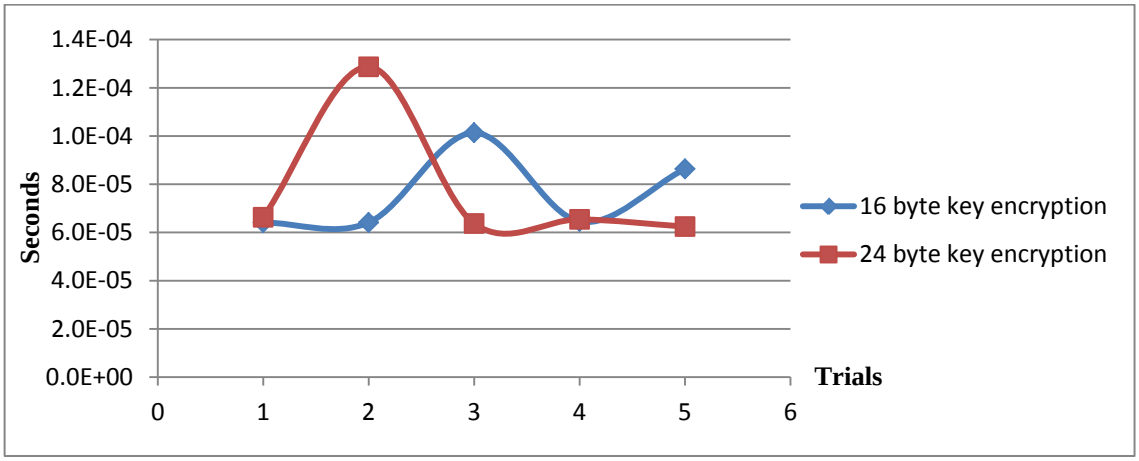

Figure 42. Encryption trials for 3DES, text size 16 with different key lengths.

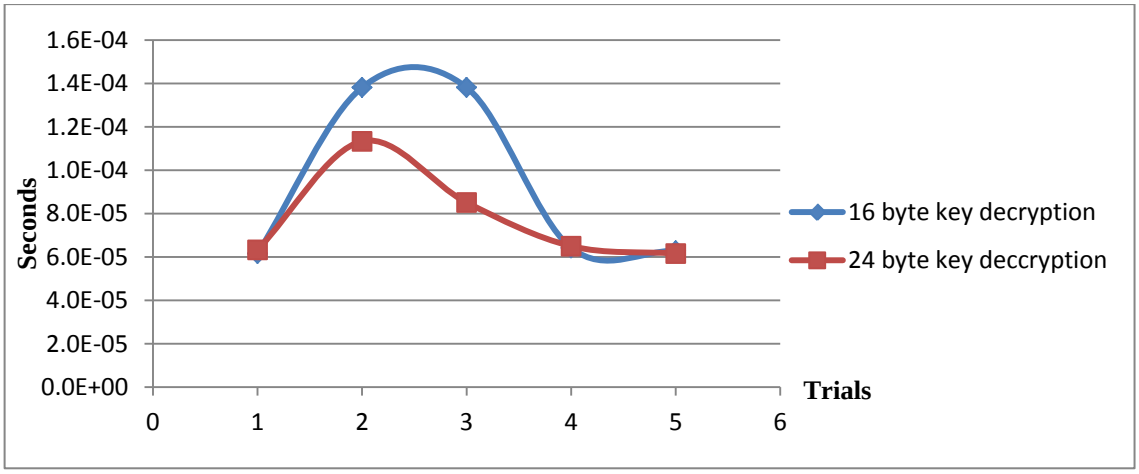

Figure 43. Decryption trials for 3DES, text size 16 with different key lengths.

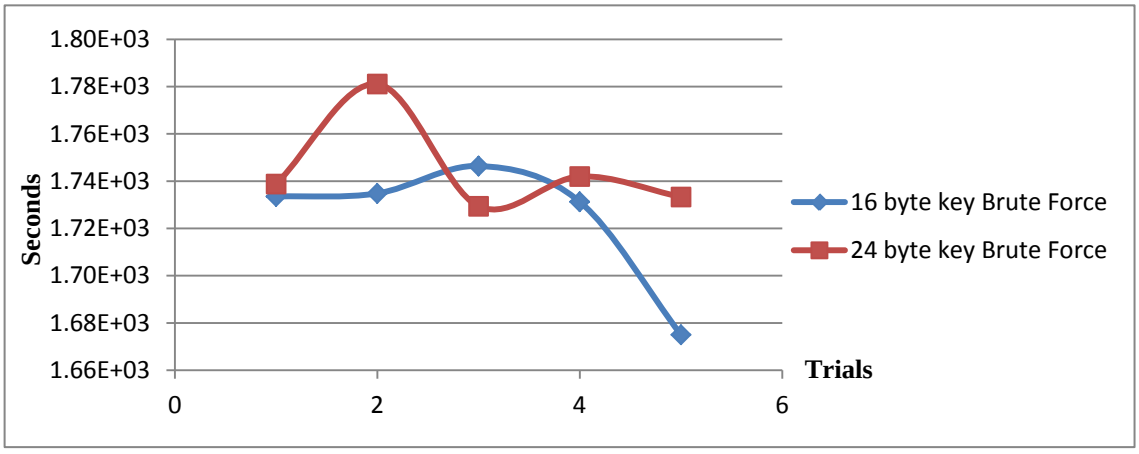

Figure 44. Brute Force attack trials for 3DES, text size 16 with different key lengths.

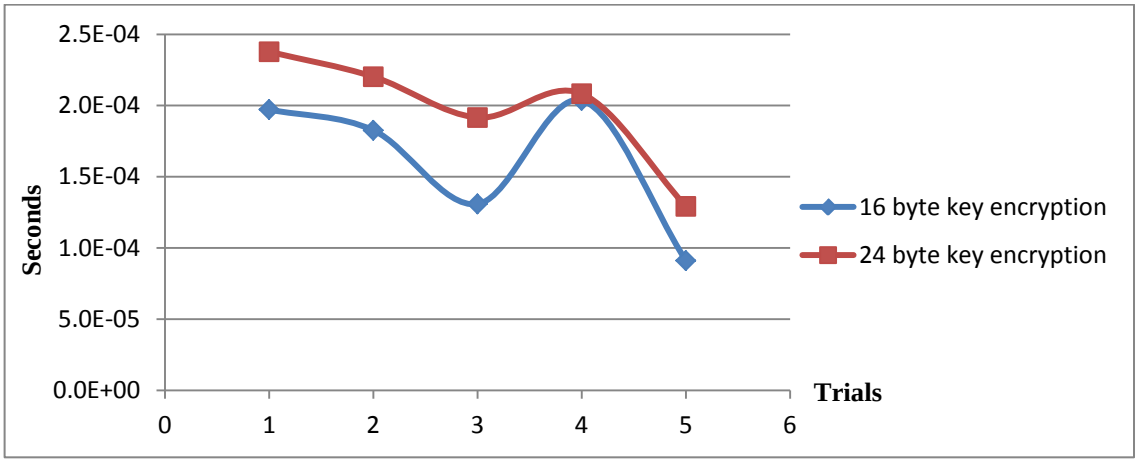

Figure 45. Encryption trials for 3DES, text size 1024 with different key lengths.

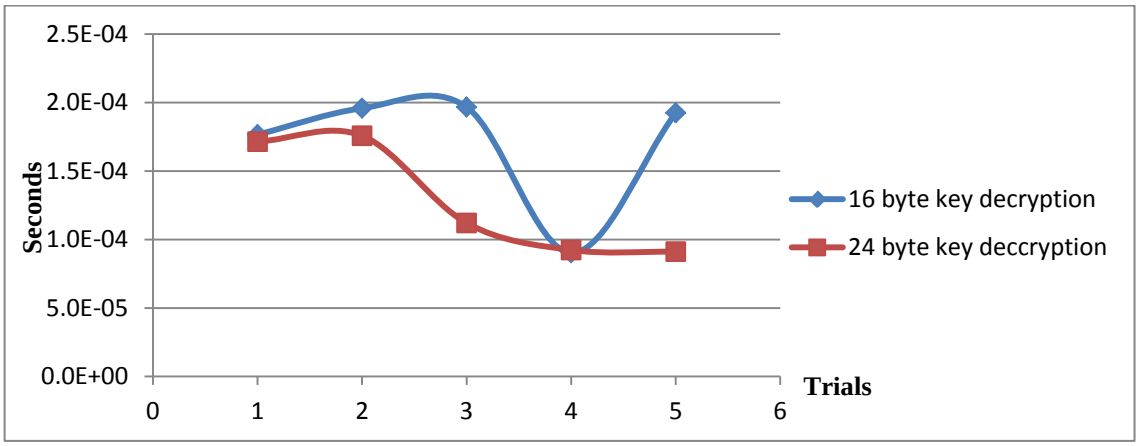

Figure 46. Decryption trials for 3DES, text size 1024 with different key lengths.

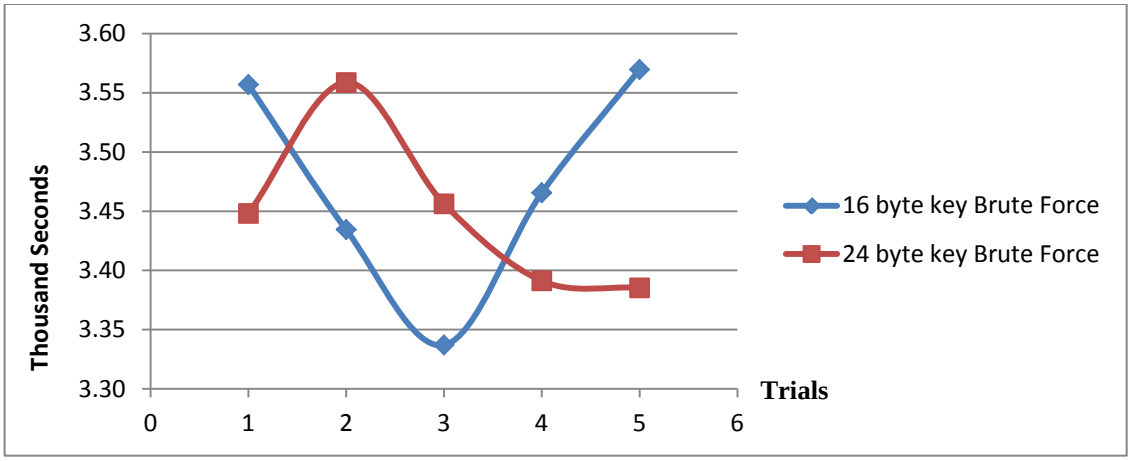

Figure 47. Brute Force attack trials for 3DES, text size 16 with different key lengths.

Figures 48, summarizes the encryption and decryption times, while Figure 49 shows the brute force attack times

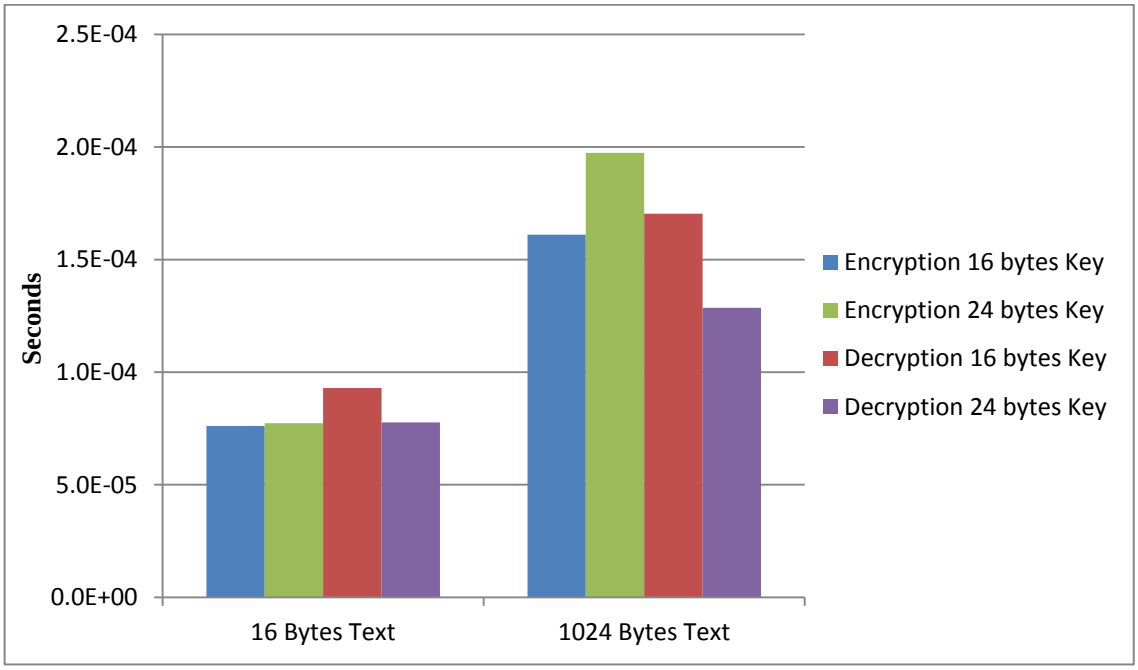

Figure 48. Encryption and decryption average time for 3DES, text size 16 and 1024 bytes with different key lengths.

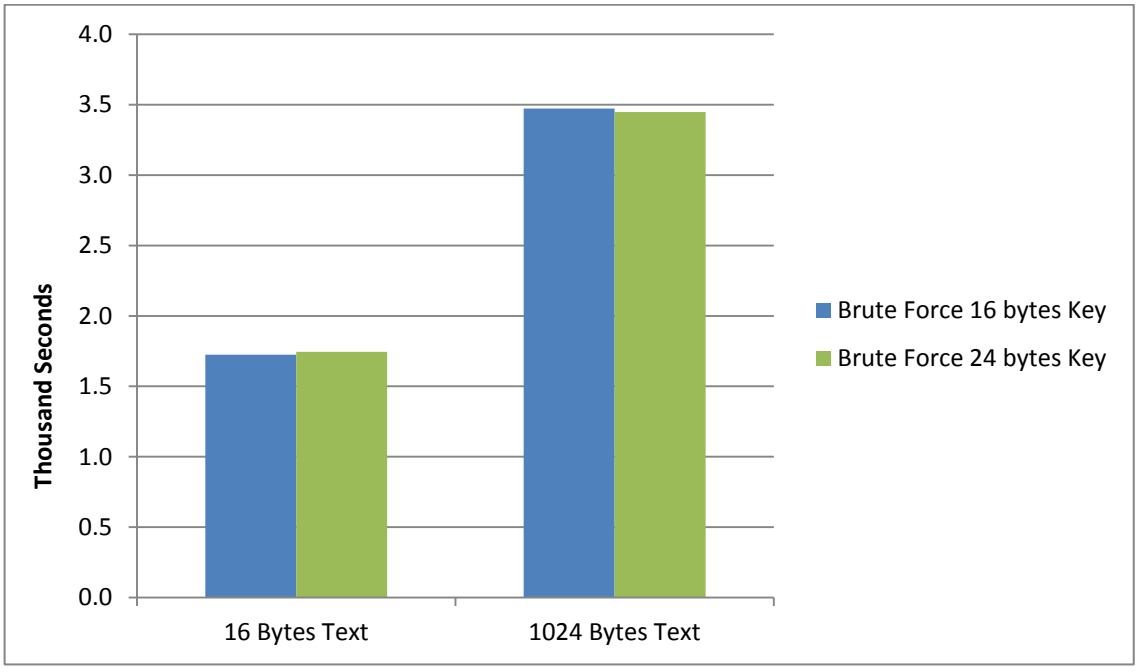

Figure 49. Brute Force attack average time for 3DES, text size 16 and 1024 bytes with different key lengths.

5.1.4. AES

Three keys were used, 16 bytes (0000000099999999), 24 bytes (000000000000000099999999) and 32 bytes Key (00000000000000000000000099999999)

The cipher texts for the 16 bytes plain text is

16 bytes key:

wDK4q4b/9obx2h3aiTd0yg==

24 bytes key:

SljuwU1LoTjXx2Imtsruqg==

32 bytes key:

cy6ikfCY3ITpKNT7s9mbdg==

Tables 14 and 15 displays encryption, decryption and brute force attack results. While Figures 50 to 55 show these times in relationship to each other’s.

Table 14. Encryption, decryption and brute force time for AES on 16 bytes text with different keys

16 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	StdDev
Encrypt	16B	3.6350E-5	9.4084E-5	9.3441E-6	9.3441E-6	3.5923E-5	3.7009E-5	3.4604E-5
	24B	3.7206E-5	1.0178E-4	3.5923E-5	3.5068E-5	3.6778E-5	4.9351E-5	2.9321E-5
	32B	3.6350E-5	7.4412E-5	7.4839E-5	3.9772E-5	3.6778E-5	5.2430E-5	2.0305E-5
Decrypt	16B	3.5923E-5	8.6386E-5	4.3193E-5	3.6778E-5	3.5068E-5	4.7469E-5	2.1990E-5
	24B	3.3785E-5	8.4248E-5	3.5495E-5	3.2929E-5	3.3785E-5	4.4048E-5	2.2491E-5
	32B	4.4904E-5	9.3228E-5	4.8752E-5	3.4640E-5	3.5923E-5	5.1489E-5	2.4079E-5
Brute Force Attack	16B	7.1090E+2	7.0100E+2	7.0048E+2	6.8354E+2	6.9876E+2	6.9894E+2	9.8328E+0
	24B	7.2169E+2	7.0812E+2	7.1690E+2	7.6204E+2	7.2928E+2	7.2761E+2	2.0724E+1
	32B	7.0765E+2	7.0546E+2	7.0697E+2	7.2328E+2	7.0933E+2	7.1054E+2	7.2545E+0

The cipher text for the 1024 bytes plain text is

16 bytes key:

zWC/iVNmmiQ5H7Ps9AXw/olW6UEP/azYRRLkWoNTEDBRX6lQIZOPUIJj1t/yIB
BXsUjXoQaeeOMVCW76i8zUcJaSHPgp8C0O7pFG/2GkMEuIBBmDrKecA0mHkW
9nd0YraAKzSEKC3RUVlBonIVSwfihDHURBS769A7b5+YhbhrnCSioez8MAQscXC
jbzhTPPPELcNszf7mtE6bKPewbZdZHkFS2HaqO4Ywm+C39pxATl0fO69owel6QI2
GgvC3jWLJmB9m0Fv87hujGnNwH0zdS8g1gNFhylz5UnWHdOMfc3IMQB2UmCtG
kEbJRDS6amDgMG5xJLijtQKH4fXcal6URZv2CpE3Yvnz+lM7VRJwOvFZhamyHqB
fKzIMArU8J+RU5ll5CSfQTJqFe1aXLuNCNOlnSX1YJZW+bgtvIFleYp55fzr6twl1fjG
VCV9IWtruYw97HHeJsjK11GUlpXt+0+ziv4EfxN8ZBLeiojdPUXN31MmpJIX6skEX
b++xVfdudO0S8BDArOdIGyoj5PLv7BX97hq823Xh6xrei7Do55r06wNFxL+TiqeN9N
DZh0kgc2PFmPyafXlStowR4sSw==

24 bytes key:

Y0kE9N/ubtssitqMWSOgN3VRkiWRy2nqosdy6fUENdUA4ER3YEbVF1cRhEcRbRV
bKh54/YtPOHEtgTLZZNsMexwwIqpATwu6KtWEB9BabZkC5AUG0W7fFOeOA1Bi
spWzKL8rSeYRhuX3/xStURTAGYFn/mFfuDSN+ghPuxdOAmqAtpxxmnWg5hsPwld
NONshDnHKF1VjabYMVdbL/qnyfecfNp919tF4+VIGrxUZT6sorF3YuFW4jAroCcGx
2F6nud8nup3kD+6zfrCDBbRjG50i7eOru29G167tYEw94ZB/1LNguISsw5YXdO8/FHi
W3lxsI4/JHjMTuKCJo4lQFYUV0Vlv2vqNfvg76S6uR8SI9eO678GDK1K491QSs+e5
RMSnDxZI7zPcjtcLJkvR+6zJzklsDUAQ1KuzLVqmk9EZih9oYz7ex+Xzvx0BmT9NO
m/EtbtDRy1tzRKmo7gARPW5tlnSM74kzspJ4eZAgwQvwXpP9uOrQlfRVp0A8lbgE
WbAfFBundXqzNGhfK8CVbz+8O24KKzOxhls72kMWw04afk0eJMAB3HAG7SEwo
qawqf/V08YQycn2J77pvQPRw==

32 bytes key:

Kj0oKoHIxriUn80oqeLDNnWLXrgLw6A9h1vHJV691qL35ulMDv0OmvsQtF1wcspu/
d8SbXmMScm/s5a/SgK3FqKhc5opf/f9ZaAYfhZTggwy3rsu+RA+wyUkfONeNVe7gH
3BalEiOUrkR4SJA5WiC4XDHMvlHK5YiI8AbeIFjc98EH2OmToqBTJfR3CyWuoLo
E0WDte+4aZmsbgwk/kKSdXwHv4RqS0TH6D5qg4xZ7q0Af+ppF2Z50LHIqS2wfr9Z
ZUG6kRYKpBqqQ9CH2JdTPVQTuOYtiqwhfuVS24zDkrGrSxcJJVBhnngIS8fzJ6D6
TulQYoghDhajRwypEc6yBNU1wdtMGRoli9X+HCK6K9LHQPlcK/NtHuRPDiMbaro
MLvsPuwhgRuEfPr9kx/iT2sGrkZVAU0LqcaDwF41b2t6UZgGnUiR/AE9Z31qEV7Dp
T2t589Dk0aHLWU90Z4hqVDBU59MP8Vx9PlLTcocLEthZ7xZP0Pi4xT9LoTNGb0R
uPIBCSdH4aKQ+JO9nBzN2m/FstP0OPrMLxCXpaS2WlJ7boAcFpvzcMICwT1vkUX
0an68eX0t4rIqLRd6bK3IFw==

Table 15. Encryption, decryption and brute force time for AES on 1024 bytes text with different keys

1024 Bytes Text	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	StdDev
Encrypt	16B	3.7206E-5	9.9643E-5	3.5495E-5	3.6778E-5	3.6778E-5	4.9180E-5	2.8217E-5
	24B	1.1932E-4	4.7897E-5	4.5759E-5	1.2958E-4	4.3193E-5	7.7149E-5	4.3362E-5
	32B	1.3300E-4	1.2659E-4	1.3172E-4	1.3599E-4	4.5331E-5	1.1453E-4	3.8830E-5
Decrypt	16B	3.6778E-5	3.4212E-5	3.6350E-5	3.5923E-5	3.4640E-5	3.5581E-5	1.1070E-6
	24B	1.1461E-4	1.1974E-4	4.2338E-5	1.1974E-4	4.1910E-5	8.7669E-5	4.1630E-5
	32B	1.5182E-4	9.9215E-5	4.6187E-5	1.1932E-4	4.0627E-5	9.1432E-5	4.7730E-5
Brute Force Attack	16B	1.1192E+3	1.1025E+3	1.0945E+3	1.0795E+3	1.0860E+3	1.0963E+3	1.5426E+1
	24B	1.1774E+3	1.1715E+3	1.1694E+3	1.1702E+3	1.1783E+3	1.1734E+3	4.1944E+0
	32B	1.1508E+3	1.1621E+3	1.1508E+3	1.1453E+3	1.1524E+3	1.1523E+3	6.1072E+0

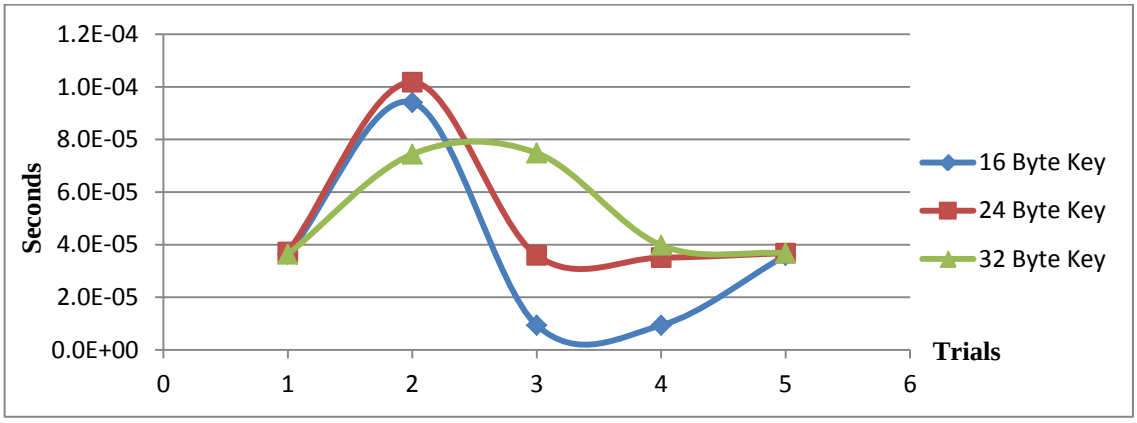

Figure 50. Encryption trials for AES, text size 16 bytes with different key lengths.

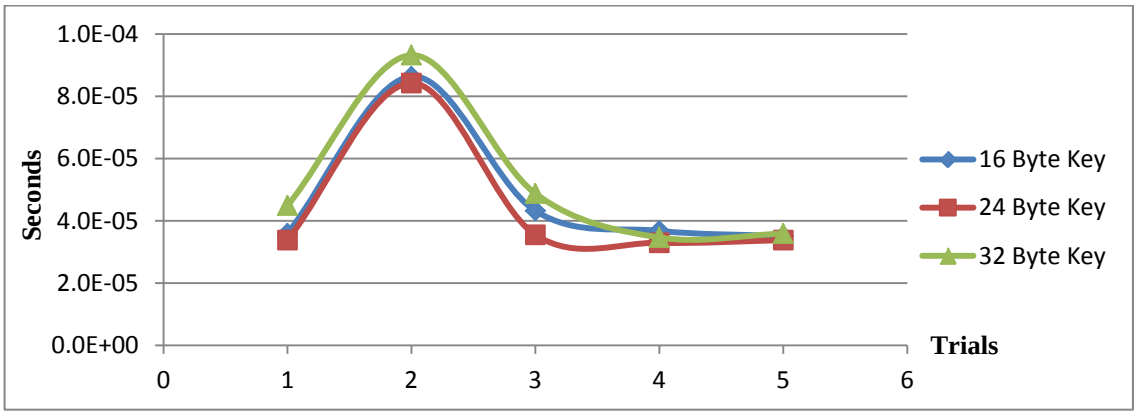

Figure 51. Decryption trials for AES, text size 16 bytes with different key lengths.

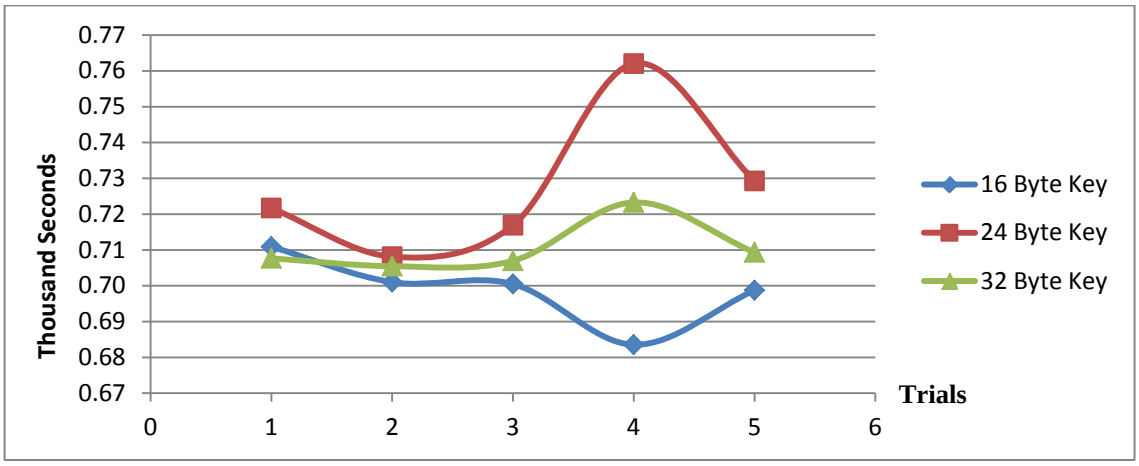

Figure 52. Brute Force attack trials for AES, text size 16 bytes with different key lengths.

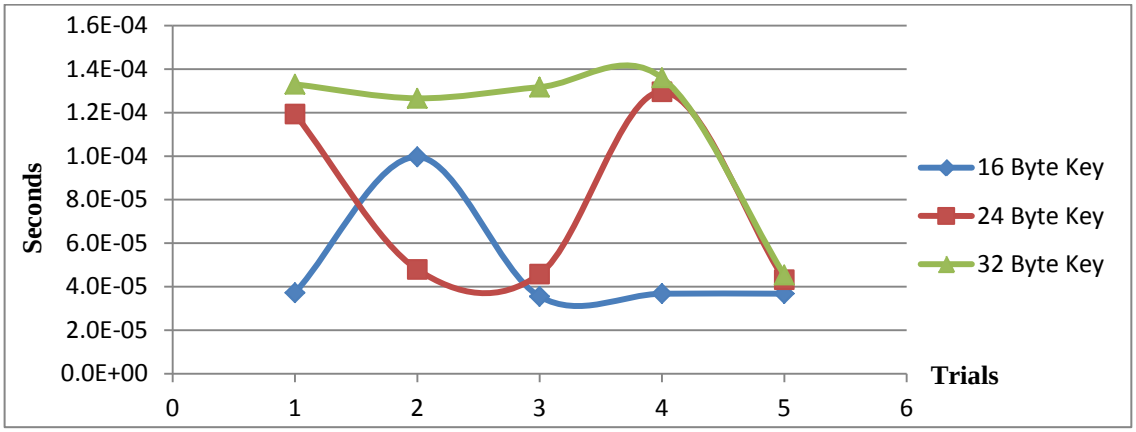

Figure 53. Encryption trials for AES, text size 1024 bytes with different key lengths.

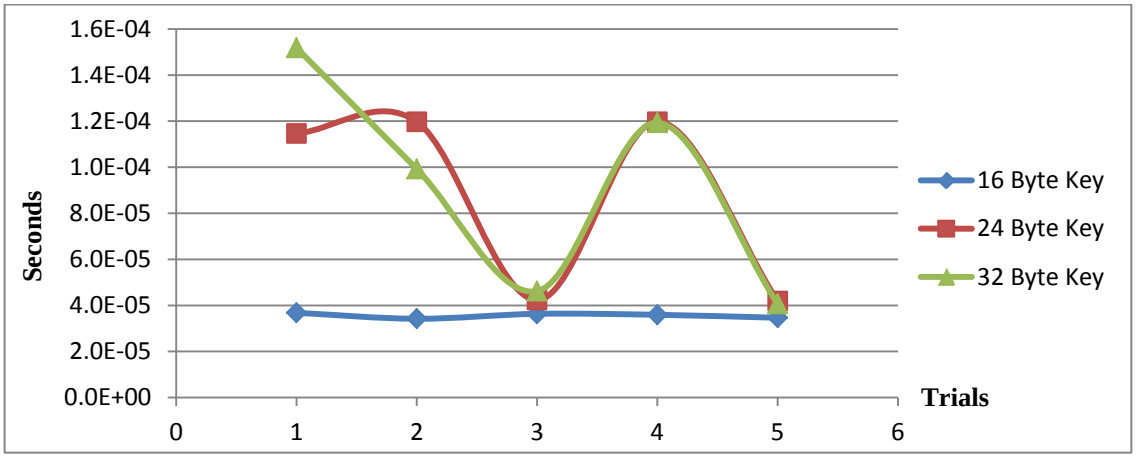

Figure 54. Decryption trials for AES, text size 1024 bytes with different key lengths.

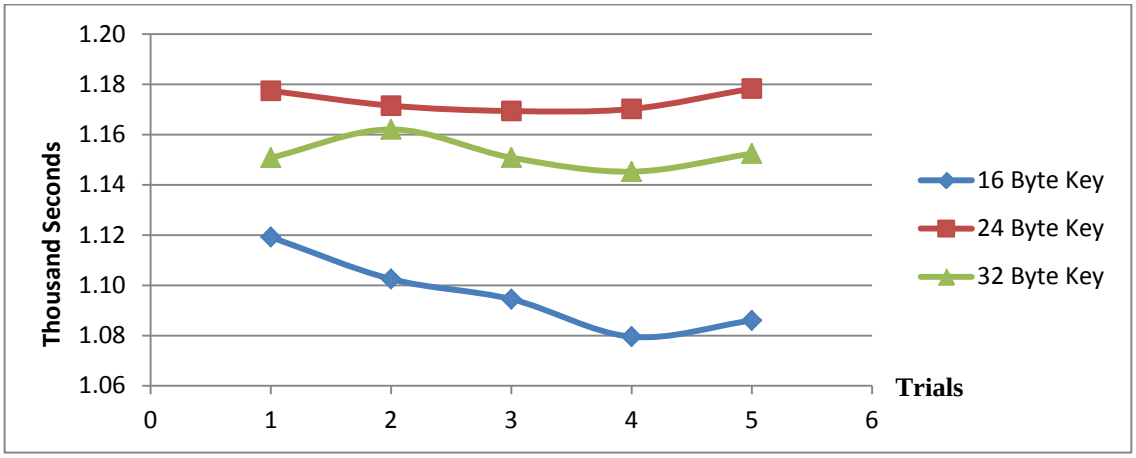

Figure 55. Brute Force attack trials for AES, text size 1024 bytes with different key lengths.

Figures 56, summarizes the encryption and decryption times, while Figure 57 shows the brute force attack times

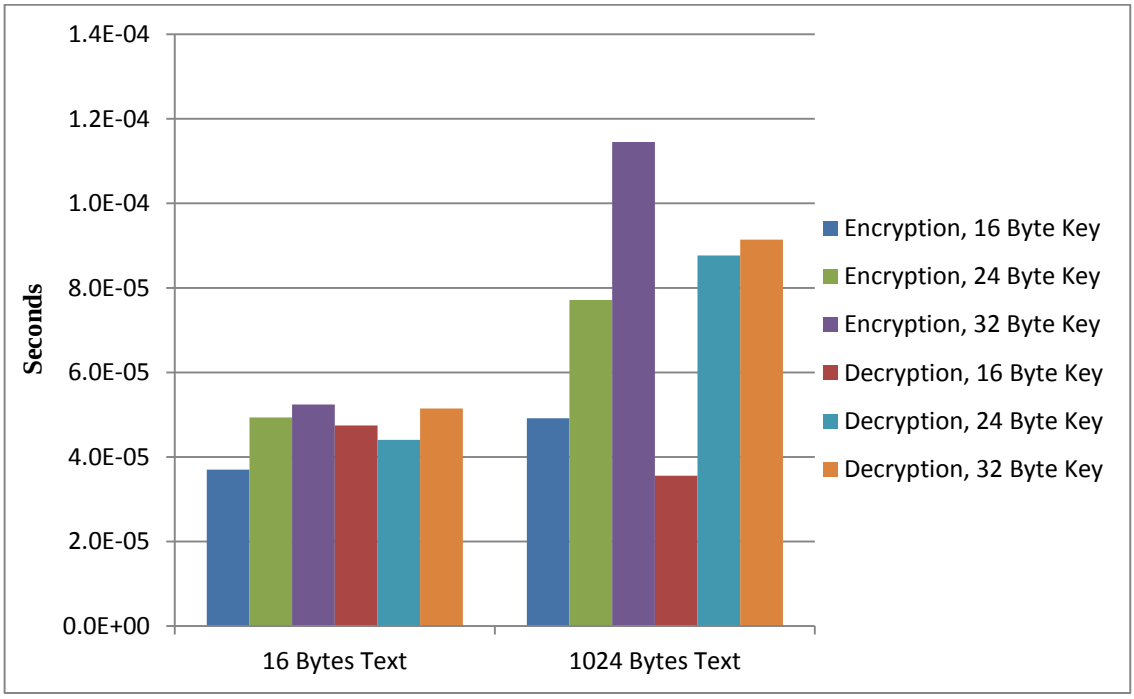

Figure 56. Encryption and decryption average time for AES, text size 16 and 1024 bytes with different key lengths.

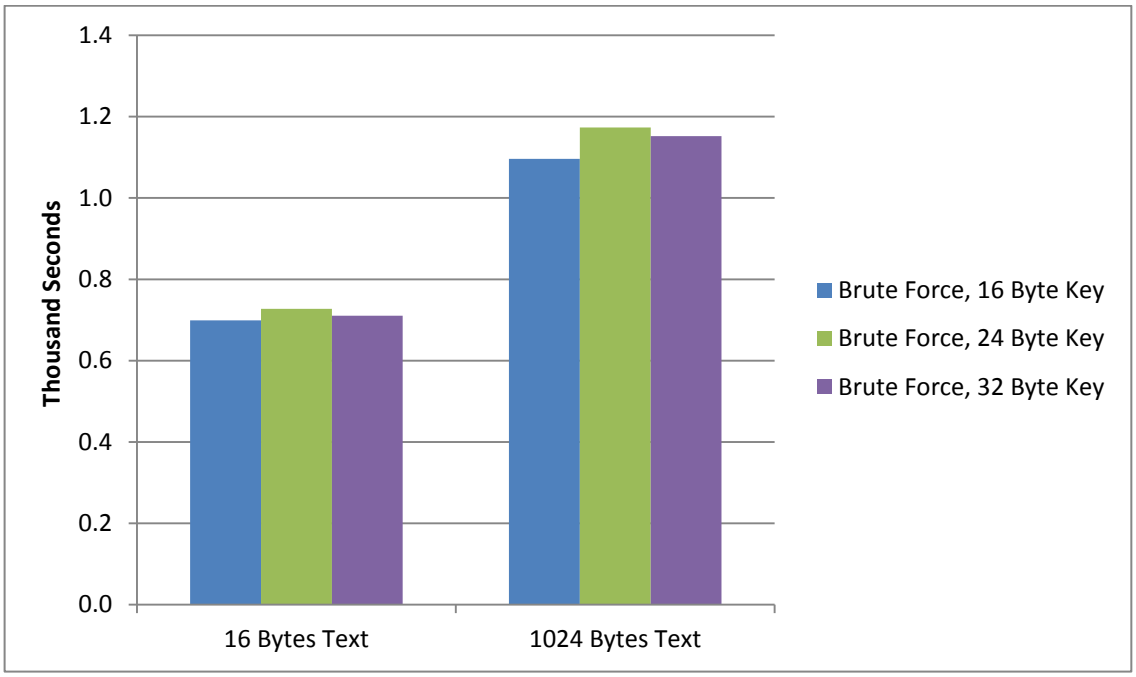

Figure 57. Brute Force attack average time for AES, text size 16 and 1024 bytes with different key lengths.

5.1.5. Blowfish

Four keys were used, 8 bytes (999999999), 16 bytes (0000000099999999), 24 bytes (000000000000000099999999) and 32 bytes Key (00000000000000000000000099999999)

The cipher texts for the 16 bytes plain text is

8 bytes key:

wyRRejFc9Yy1z1IfDaRZJg==

16 bytes key:

DeYwqctCGMsQQIPGzR96Ww==

24 bytes key:

Pswz+4rotXkJkwgiFxe7oA==

32 bytes key:

zBt04o6V+444fWI7IB8UrQ==

Tables 16 and 17 displays encryption, decryption and brute force attack results. While Figures 58 to 63 show these times in relationship to each other's.

Table 16. Encryption, decryption and brute force time for AES on 16 bytes text with

16 Bytes	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	StdDev
Encrypt	8B	1.2659E-4	6.8852E-5	6.8424E-5	6.9280E-5	7.6978E-5	8.2024E-5	2.5160E-5
	16B	6.9707E-5	1.2872E-4	9.1945E-5	7.0135E-5	6.8852E-5	8.5873E-5	2.5845E-5
	24B	7.0990E-5	7.0135E-5	8.8524E-5	7.0563E-5	7.0990E-5	7.4241E-5	7.9926E-6
	32B	7.0135E-5	7.8260E-5	6.6714E-5	7.0990E-5	1.0349E-4	7.7918E-5	1.4902E-5
Decrypt	8B	6.8424E-5	1.3129E-4	6.7569E-5	6.9280E-5	6.5859E-5	8.0484E-5	2.8429E-5
	16B	6.5431E-5	1.1718E-4	6.7997E-5	6.7569E-5	6.8424E-5	7.7320E-5	2.2311E-5
	24B	1.0306E-4	1.6123E-4	6.7997E-5	6.6714E-5	6.6714E-5	9.3143E-5	4.1119E-5
	32B	6.9280E-5	6.6286E-5	1.2017E-4	9.0662E-5	6.8424E-5	8.2965E-5	2.3024E-5
Brute Force Attack	8B	4.2154E+3	4.0620E+3	4.0671E+3	4.0477E+3	4.2266E+3	4.1238E+3	8.9157E+1
	16B	4.1933E+3	4.2197E+3	4.2157E+3	4.0569E+3	4.0494E+3	4.1470E+3	8.6298E+1
	24B	4.0835E+3	4.2635E+3	4.2534E+3	4.2655E+3	4.2619E+3	4.2256E+3	7.9562E+1
	32B	4.2441E+3	4.2535E+3	4.2335E+3	4.2665E+3	4.2759E+3	4.2547E+3	1.6978E+1

The cipher texts for the 1024 bytes plain text is

8 bytes key:

33bhZbLL4sz+RtER2tmMV2Orw4i5zlCr20lLhw1qmk1YFxO4tcWpTuLr0T2QOEmq
ZZeLwOz7L5HNHMhjU4eQZ1UMH5+ypSqYBKDm8LWZ0xGhfOtNeoCqzLiH0wY
ePx7wsMI+c7lbayykYTpZ+AfhIiaQnvwwhTw+XKv7c2ktYvWOow6rREp55sK5Ysbk
6fu5HnzdU8nrkTLqC9IVqbc9qp0Ttjarq8hvXyuTM6NJi6YwAOAhs5KQiVhEXAvn4
EzFXk/xVo1kJhT8JF8iyLi1+QFUdTZK0rkL7MMAAON6qbEoXH9Qf7PVF6sW57h5
hki+BYCYGQ6261X3bqsa/qdthIeOt7lMDsPlkz/5Ebr1daEoXH9Qf7PVF/lYmuoyRky

K2pRXVL2XdrioNufEJ92N+tvP5pbvzx32XRFEsqm+Mwfyv7o9g0FjJW5YTnG7CMI
LfoFuDAiin5lkhB9PfQNksJpMtAXGmZ5Cx5O11JM0thxbxb8WRRnuM+TlkA52895
01o0MFyOvN2ijhKVrXyjUTK041rfZw57Nn10wLBZNGxD90sZH86BMkb6Pp/swvr7
4h8+ORPPdz6eQ750Kbfp4TA==

16 bytes key:

GOBFgaym3wYkmZU430AoyCuWSYihIODiUbxlZf88Ysb+FcHzWVOHjIFDSUf8sa
IW+I6YdvGGZoPSbCdn4Bl00BwRrgPqLJxgN9sDBOjE+jnqTyMKH77htMVIpIVbjj0
QRHpsEl5WIp4rruFU1I+Qz+neUUXip80bekw/5fvWEIC/zxFEaL49e6JQKT6lGj2Y8
YvgTAmzRHdUz3lSl4qbCOeC+6oFeS3ZAE01boCzXmqtLA2k8d655PTNwcbCvzP6
S3YLfNTiONSfPWByOTJY6LFDeBEXvVwTp+iLPRU/pNcJiYG/1T5qrqioT8d3vnp7
Z1Wuriwx9fGRwea0QfHGSXhAk2OrO1f7RuipTUzOG00JiYG/1T5qroZztUdbhOq9Sj
t5GUjF18NY8dR88Bwbf5esFzDpR6U2MCcQMjAIhFPhvC3Bv5vJGS8odDVuMJKH
o6IOKbd1VnDXK7cr+juBojaKmRvDwwfI2rwNcVyS3o+GmTxwvTbwj/TyawFpkxpq
H8wz7iu3t9nqW60ojf5N+2NF1brSQvw3yCZ11eyBLk9OxKWXDvDkzT6ehxYMUCz
6bnGhN+Cig7srDxy53LUr8w==

24 bytes key:

Dw3vfDqI8KImvut/+lYnOzaYzEZCb12HQPe/ZAEWSrbu1v2kJYUr80yQwtyL0EuGF
1XAPnIdMrrLDTei16lkdM3SeutLGRLhvcKolK0VPWvQEL1vG52gHFwTt7Ex340UF
NeHQCQ7GMaMs2LllutSYo2f4gsuW/nQrw1cjajwRo7LeLIz1Rso3bv6TBCTz2qVN1
5Rw2M52tdMs5z9a7/bJj3E2YlRu3s+/NpHJ4WJmp8BDQyxQqgHb7EYvl0vYEgRmT
wnsOP5kYEZRzcGyLALhH3glbqdEpXEPzyZOo9vBtwmAl+0ywYVJcre3L2CksUZ5
RerYwUusNW4HKPXPn+jaxzUXef/irKL4cQl1p9AYKUmAl+0ywYVJYfWQ8UYKh
lXT/vdPd+6JoWIz8K3Zvu4X63Q4BX1KMLUxnsafbAoaUiK6qQU9lIYkroRRVGZqp
v/oZGiXW6Xba1Z5IWdpD7SPPT2WYWZzdxxOIKRvMkhO+oVgpG+arrGxh14koha
RD/o+Udrq4U4wNVOXa6gmytowqziO95Pe3p9C4BsOsuH/lbnMCbTr/6aYqgYtErSy
Rm44qG+BIhAOHr/7obh20VehA==

32 bytes key:

3k8ZEI1TiFArEXX5aN9I5RvcGERSJ3KBcc51zDK08vF5yUNeRMXCSU2gPwybNSr
3rRStrj+Sb+I4amOlJfCZYDwZT9XiO0tTrDm+gn65/q/Fde5APp+TLMaYIfCflznUe1L

3jSS6DpWcUILgyis43q1K39FJaMS8U8+SAdojD/wyDFIhBENMnh8Gfobexk/tsvKTa
Pv83YYyHTPhzuvjQtccsWkNja7S6jAtFi+W6hHHmmxTAW2H3QbiDLDdNfgMS1p
RrE6BO3bmRp6db9+BYj5DF1HC8NkZtAlLcYnAkmTX2Oa2nUX3IwcLpJ16bd8dU
UceJ0PlNrJIFoSfV9GJgS3Kj4/eAKS7xg3BRhHDig7X2Oa2nUX3I5Fx6HMwT/iQu8k
U8E2LXIYm9fv/7MjWl+ZWED8c+JKWiBpyXQuUm2QLVnvwD+nBLIHOSM36TIy
ACBO/h6sqM5sekAYi9i8x5ZBvOe/E5GqNqIkp0CxhFz3bojs1GG3OoIS6WOyBw18L
1uTj+8/pp5ClqI0ZctQzteIq4QJyY7++gqsifjY2s2YXT1k3dPmqytDBGCv37tBHSOng+
n5NsnKazG3/7hntlw==

Table 17. Encryption, decryption and brute force time for AES on 1024 bytes text with

1024 Bytes	Key Size	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	StdDev
Encrypt	8B	1.5224E-4	1.5267E-4	1.4284E-4	1.5310E-4	1.3899E-4	1.4797E-4	6.5906E-6
	16B	1.6465E-4	1.4412E-4	1.4626E-4	1.4711E-4	1.3899E-4	1.4822E-4	9.7078E-6
	24B	1.7277E-4	1.3941E-4	9.8360E-5	1.6037E-4	7.8260E-5	1.2984E-4	4.0368E-5
	32B	1.4284E-4	1.3642E-4	7.7833E-5	1.4326E-4	8.2109E-5	1.1649E-4	3.3484E-5
Decrypt	8B	1.5567E-4	1.3770E-4	7.5267E-5	7.4412E-5	7.5267E-5	1.0366E-4	3.9785E-5
	16B	1.4626E-4	1.2830E-4	1.4540E-4	7.5267E-5	1.2231E-4	1.2351E-4	2.8933E-5
	24B	1.4711E-4	1.5053E-4	8.1682E-5	7.3556E-5	7.4412E-5	1.0546E-4	3.9730E-5
	32B	1.4497E-4	1.7405E-4	1.3257E-4	1.4925E-4	1.4540E-4	1.4925E-4	1.5219E-5
Brute Force Attack	8B	4.7120E+3	4.7290E+3	4.5738E+3	4.6334E+3	4.6419E+3	4.6580E+3	6.3054E+1
	16B	4.7072E+3	4.6979E+3	4.7228E+3	4.6187E+3	4.6543E+3	4.6802E+3	4.2764E+1
	24B	4.8397E+3	4.8007E+3	4.6669E+3	4.7148E+3	4.6264E+3	4.7297E+3	8.9419E+1
	32B	4.8605E+3	4.8630E+3	4.7852E+3	4.6898E+3	4.6704E+3	4.7738E+3	9.1331E+1

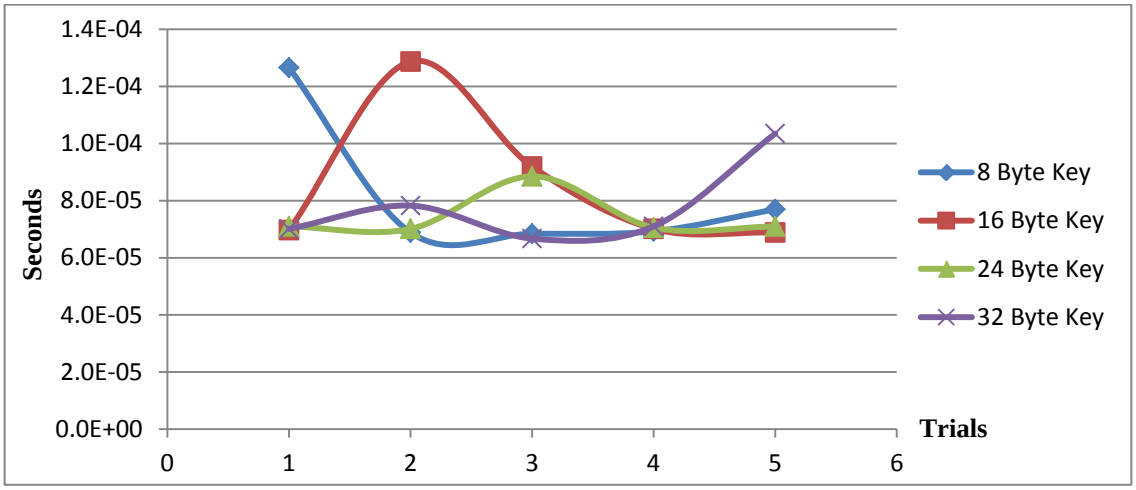

Figure 58. Encryption trials for Blowfish, text size 16 bytes with different key lengths.

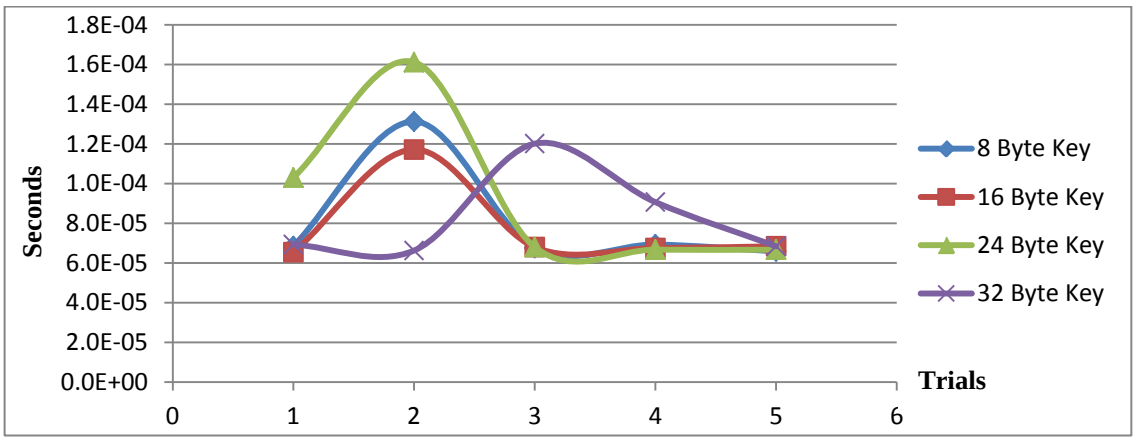

Figure 59. Decryption trials for Blowfish, text size 16 bytes with different key lengths.

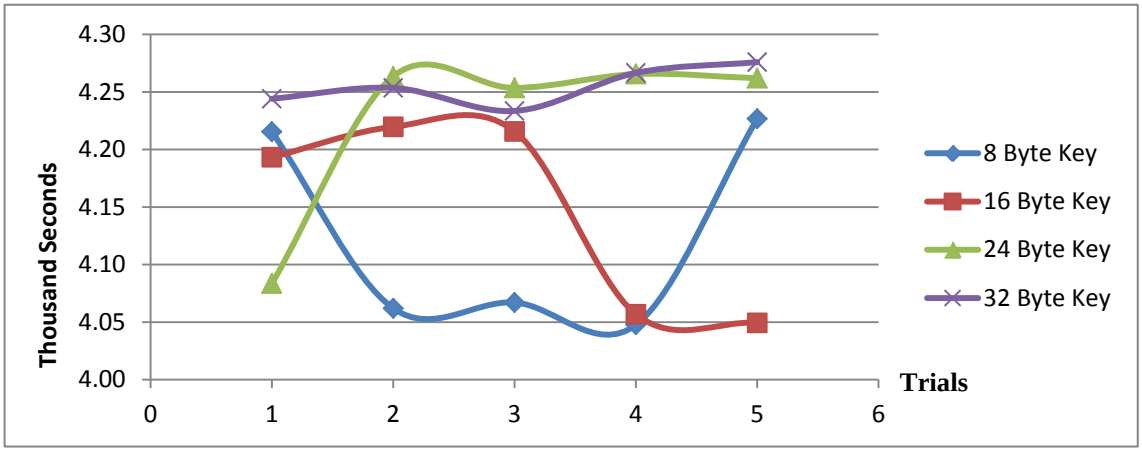

Figure 60. Brute Force attack trials for Blowfish, text size 16 bytes with different key lengths.

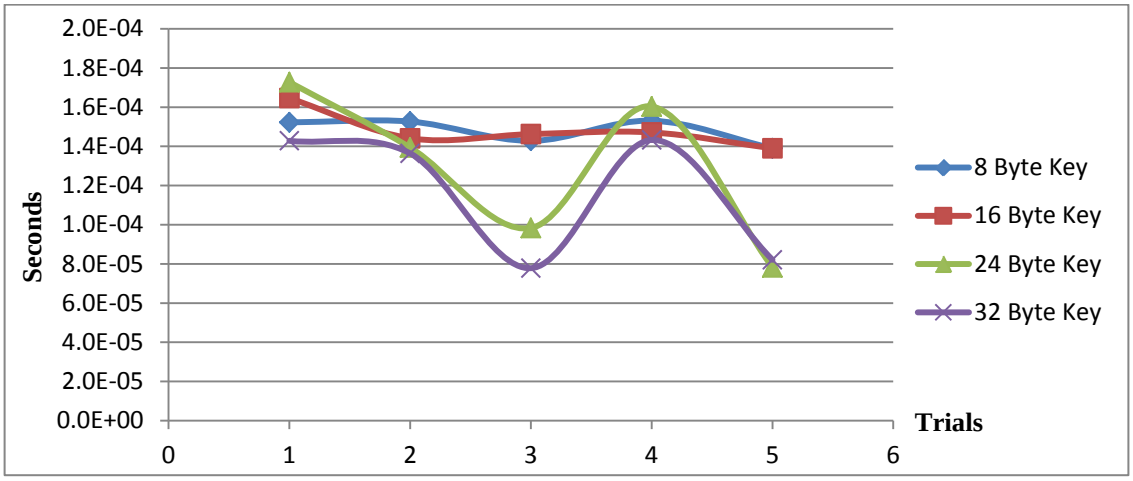

Figure 61. Encryption trials for Blowfish, text size 1024 bytes with different key lengths.

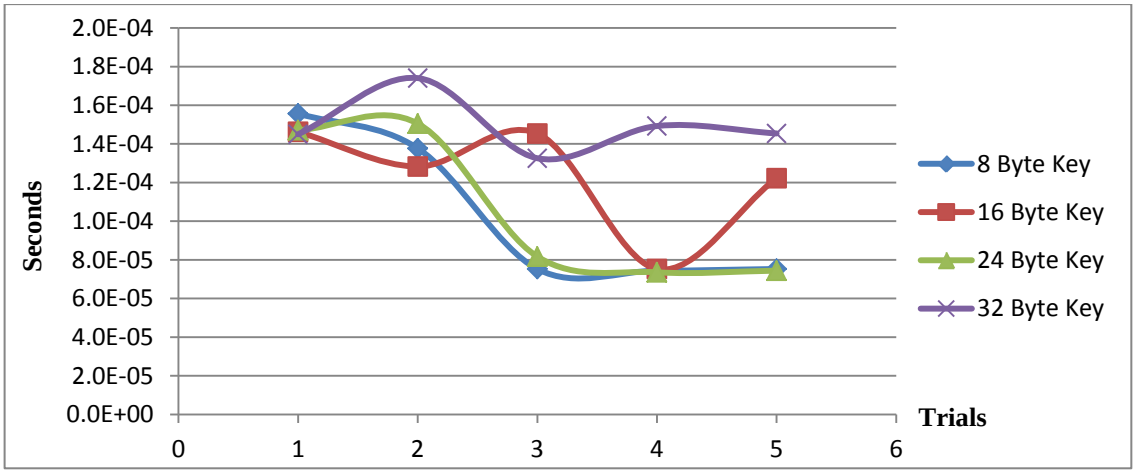

Figure 62. Decryption trials for Blowfish, text size 1024 bytes with different key lengths.

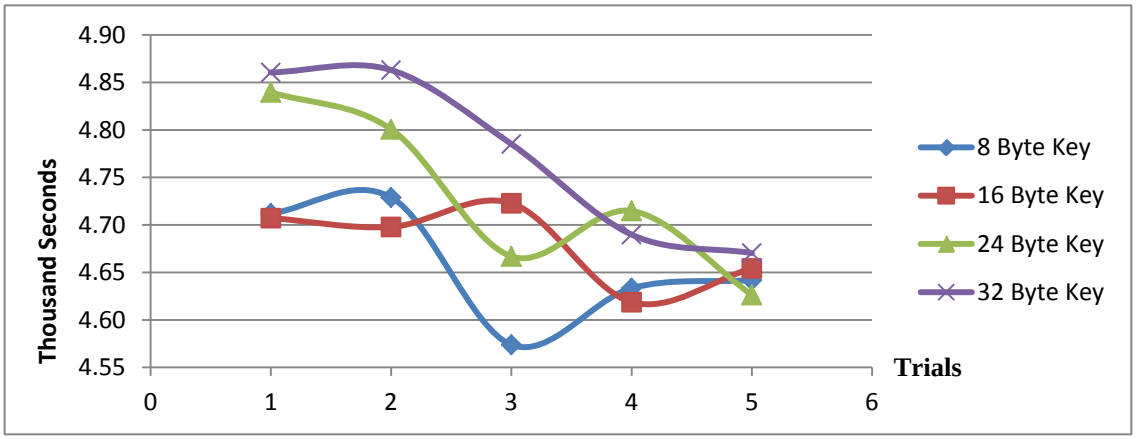

Figure 63. Brute Force attack trials for Blowfish, text size 1024 bytes with different key lengths.

Figures 64, summarizes the encryption and decryption times, while Figure 65 shows the brute force attack times

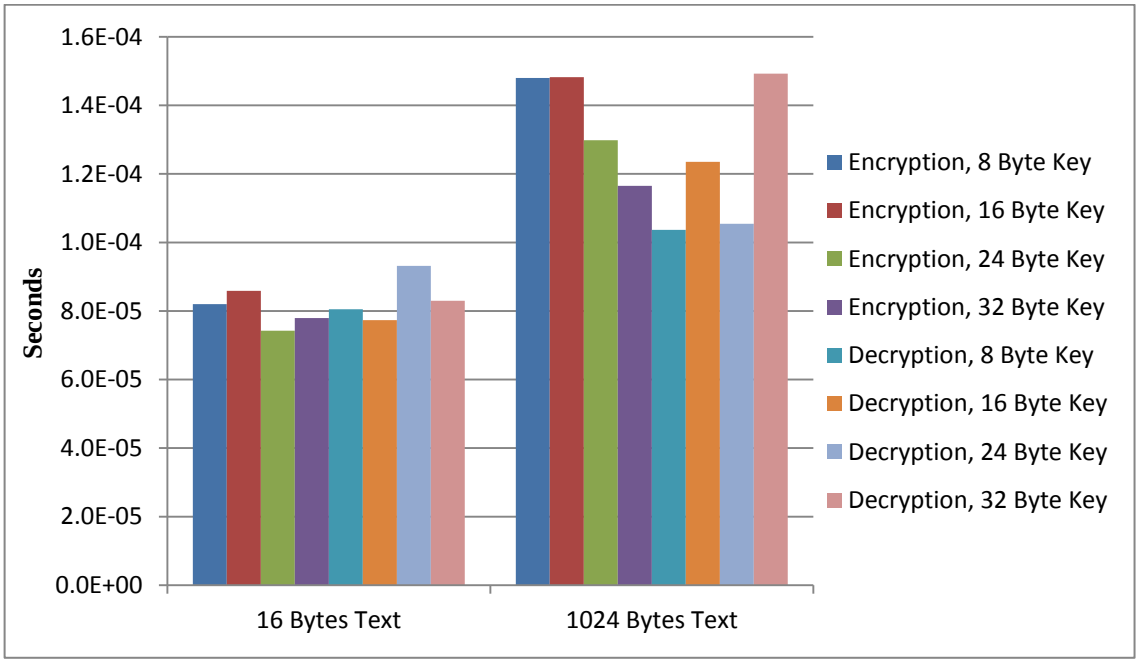

Figure 64. Encryption and decryption average time for Blowfish, text size 16 and 1024 bytes with different key lengths.

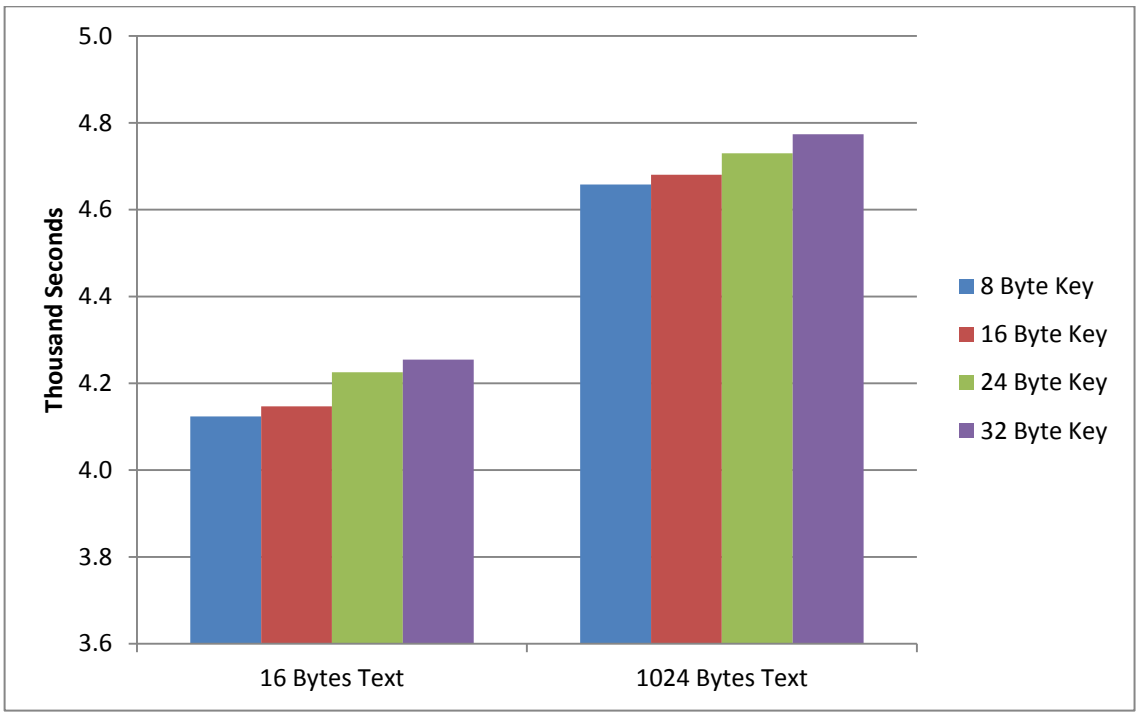

Figure 65. Brute Force attack average time for Blowfish, text size 16 and 1024 bytes with different key lengths.

Figure 65, 66, and 67 shows a general view for the average time of encryption, decryption and brute force attack times for all the algorithms with different key lengths for texts sizes 16 and 1024 respectively.

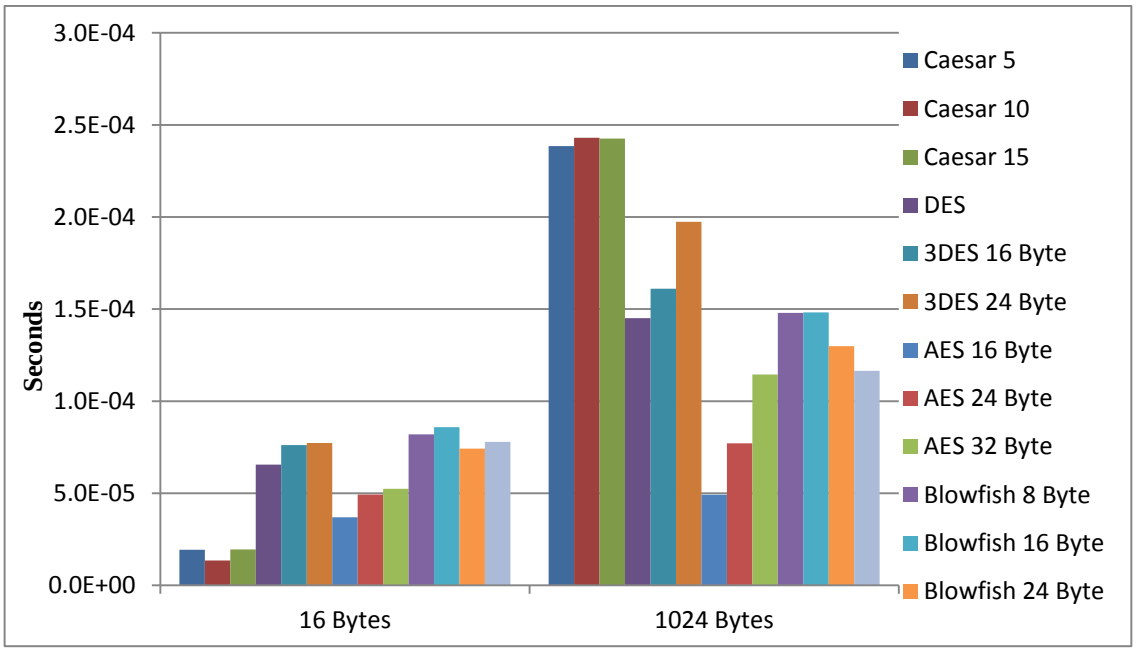

Figure 66. Average encryption time for all algorithms

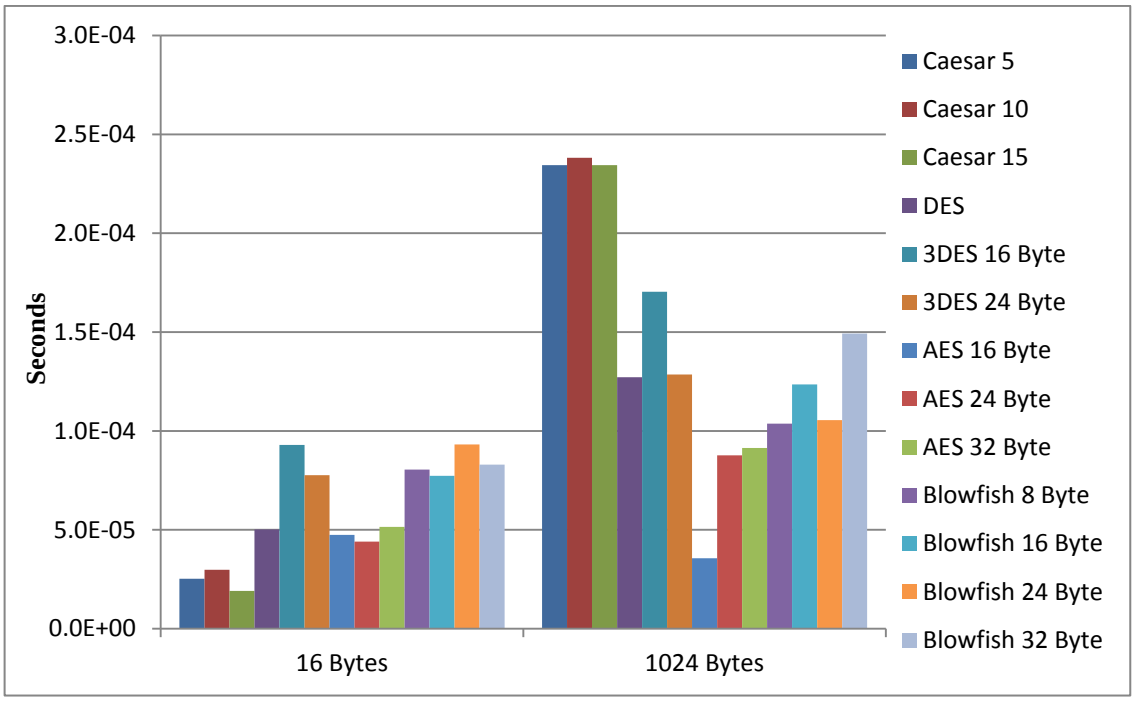

Figure 67. Average decryption time for all algorithms

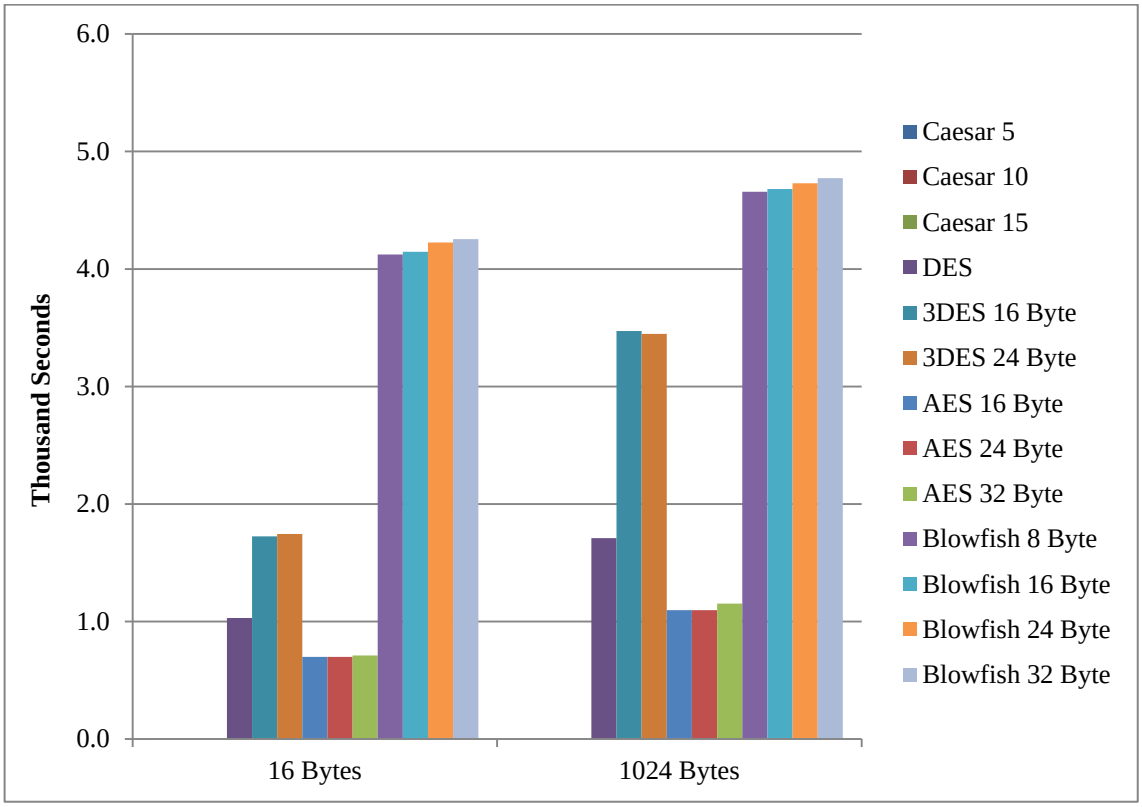

Figure 68. Average Brute force attack time for all algorithms

5.2. Phase Two Results

Phase two results are obtained from running the algorithms on the Raspberry Pi

Due to the similar architecture of the Raspberry Pi and modern mobile phones, the Raspberry Pi can be considered as an entry level smart phone. Hence, the brute force attack trials will not be conducted, encryption and decryption times and power consumption are more important in a mobile application.

The power consumption reading was taken using a multi-meter connected in series to the general purpose input/output pins (GPIO pins). The readings were taken during the idle state (running the graphical user interface, and from the command line) and while running the algorithms, the difference between the two reading is the amount of currents drawn by the device to execute the algorithms. The idle state gave an average reading of 355mA, and for all the algorithms the average current reading was 400mA, with maximum current value of 419mA, the CPU usage reached 100% when executing the algorithms, these readings was consistent for algorithms running in the terminal in the graphical user interface and in the command line environment.

Time trials results of for encryption and decryption are displayed in Table 17.

Table 18. Average encryption and decryption time of 16 byte text on a Raspberry Pi

Algorithms	Key Size	Encryption Average (seconds)	Decryption Average (seconds)
Caesar	5 shifts	0.0011	0.0012
	10 shifts	0.0013	0.0013
	15 shifts	0.001	0.0008
DES	8 Bytes	0.00165	0.0017
3DES	16 Bytes	0.0016	0.0017
	24 Bytes	0.0018	0.00185
AES	16 Bytes	0.0015	0.0015
	24 Bytes	0.0015	0.00175
	32 Bytes	0.0015	0.0015
Blowfish	8 Bytes	0.0019	0.00185
	16 Bytes	0.0018	0.00165
	24 Bytes	0.00165	0.00185
	32 Bytes	0.0019	0.0018

These results were recorded using an oscilloscope, since the Raspberry Pi does not has a precise timer function. Figure 69 shows a snapshot of the oscilloscope recording the encryption time for the DES algorithm

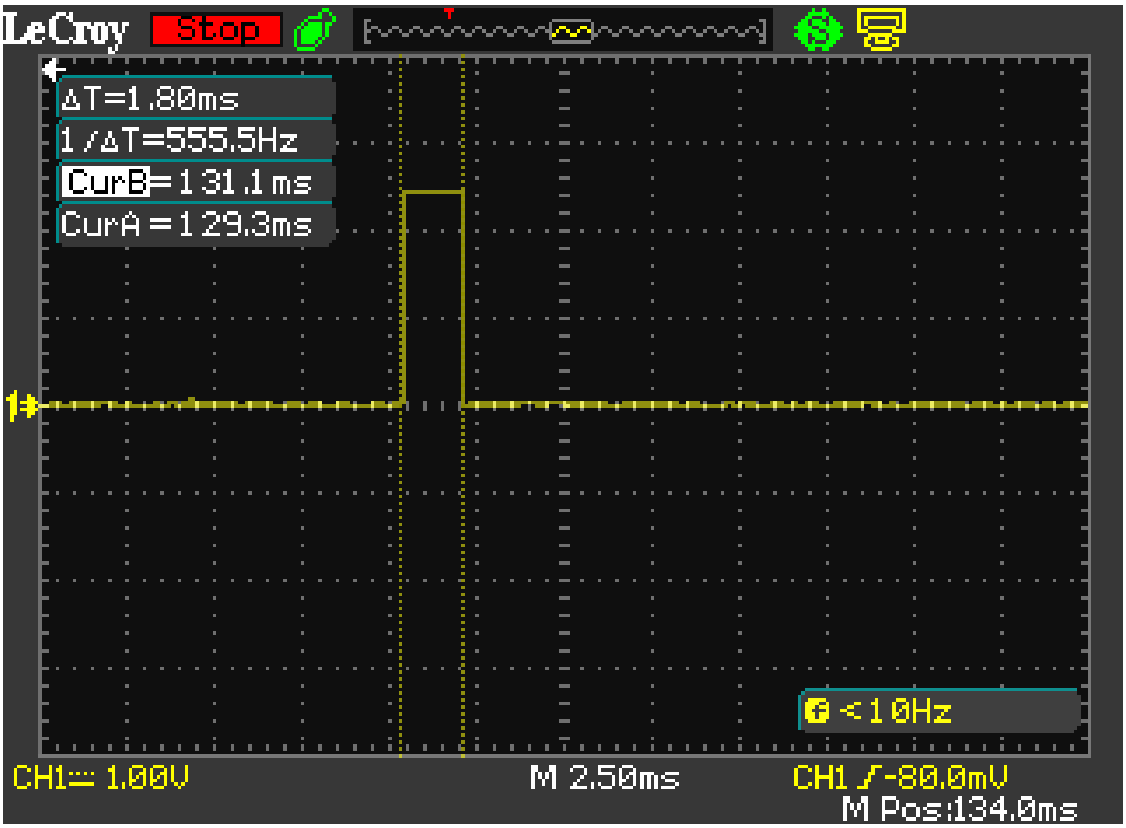

Figure 69. A snapshot form the oscilloscope showing the encryption time of DES on the Raspberry Pi.

5.3. Discussion

The discussion will be separated into parts according to the encryption algorithms.

There are several points to be mentioned regarding the testing process and the obtained results:

- Phase one tests were conducted on a general purpose laptop computer, with preloaded operating system and different programs running in the background, it was not possible to obtain identical testing conditions.
- The different programs running at the same time yields a more accurate results, since some of the NFC devices has an operating system running on them, with background processes running while the device is either encrypting or decrypting data.
- The reduced key space is important for two reasons
    - It allows obtaining results with the given time
    - It simplified the brute force attack algorithm.
- Due to time constrains, it was not possible to run the tests more than five times per algorithm, the average time for running the brute force attack on the windows computer was 271406.678170143 seconds which is equal roughly to 75 hours of consecutive non interrupted running time.

Caesar: Caesar encryption algorithms is significantly affected by the text size, for the small text, Caesar had the fastest in encryption and decryption times, however with the larger text size, it had the slowest encryption and decryption time. The brute force attack times were consistently the shortest of all the encryption algorithms, by a significant margin.

DES: DES key is 8 bytes long, with parity bits at the end of each byte, this reduces the total possible key space, in the brute force attack test that was conducted. The used key for encryption was (99999999). However, the key that was used to successfully decrypt the cipher text was (88888888). This can be explained in Table 19.

Table 19. comparison between encryption key and brute force attack key.

Cipher Key	0000100**01**	0000100**01**	0000100**01**	0000100**01**	0000100**01**	0000100**01**	0000100**01**	0000100**01**
Used Key	0000100**X**	0000100**X**	0000100**X**	0000100**X**	0000100**X**	0000100**X**	0000100**X**	0000100**X**
Brute Force Key	0000100**00**	0000100**00**	0000100**00**	0000100**00**	0000100**00**	0000100**00**	0000100**00**	0000100**00**

This property of DES significantly reduces the available key space from 2^{64} to 2^{56}. And this not a specific case for the reduced key space, it is true for all possible key space used for DES algorithm.

3DES: this algorithm is an improvement over the standard DES, as seen in the results, the brute force time is improved over the time needed to break DES given the same amount of trials. There is an increase in encryption and decryption times over standard DES. However, the increase in encryption and decryption times when compared to the increase in the needed time to brute force an encrypted text is smaller. The key used for the successful brute force attack was identical to the key used for encryption. 3DES consists of three DES phases, for encrypting, it encrypts with the first key, decrypts with the second key, and encrypts again with the third key. And the decryption process is the executed in a reverse order. For a key length of 16 bytes, the first and third key are the same, while in 24 bytes key length, each key is different, and that is why it is not possible to expand the key to more than 24 bytes.

AES: except for Caesar, AES is the fastest algorithms for encryption and decryption even when using the longest key length possible, however this speed is also reflected on the brute force attack time. The AES encryption and decryption time is significantly affected by the plain text size, for the 16 bytes text, the encryption and decryption time differences were very small, these differences increased with the increase of the plain text size. The text size had no noticeable effect one the brute force attack time with different key sizes.

Blowfish: encryption times for the different key lengths had almost an inverse relation, the smaller the key, the longer it takes to encrypt, this is true for both text sizes tested,

the relation between decryption times and key lengths are different, with smaller texts, the decryption times were close to each other, and with the larger text size, the largest key length had the largest decryption time. The inconsistent results can be explained by the status of the machine running the algorithms with different programs running in the background, the difference between the longest and shortest decryption times for the small text however is 15 micro seconds, and for the larger text the difference is 45 micro seconds

The encryption and decryption times obtained from the Raspberry Pi have a much smaller difference between different algorithms (maximum difference for encryption times was 0.0008 seconds and for decryption times 0.00105 seconds), compared to the average encryption and decryption times.

6. Conclusion

Choosing the suitable encryption algorithm for any application is subject to several factors, the level of security needed, the amount of available memory, the available time for encryption and decryption and many others.

Based on the conducted tests and obtained results, several conclusions can be made about all the test encryption algorithms.

The Caesar encryption algorithm has the worst performance out of the tested algorithms, it had the longest encryption and decryption time, and the least immunity against the brute force attack, also it does not change the basic structure of the text, does not hide the frequency of the symbols in the text, and since it is based on substitution, it does not have a key. The Caesar algorithm is not suitable as a modern encryption algorithm, no matter what application it is used for. It does however provide a convenient way for cryptography and help familiarize some basic concepts for educational purposes.

Based on the results DES is more secure against brute force attack than AES, given the same number of trials, which is DES major weakness, the limited key space makes calculating the key an easier task, which is helped by the reduction in the key space due to the exclusion of the parity bits.

AES security is based on the significantly larger key space compared to DES. For the same key used for both algorithms (setting the rest of the AES bytes in the key to zero) AES encrypted text can deciphered faster than a text encrypted with DES, however, this can be mitigated using a longer key, for each byte, the brute force attack time needed to decipher an encrypted text is multiplied approximately by ten times.

The short encryption and decryption time of the AES is another important advantage, from the current consumption readings, a shorter encryption and decryption time means less energy used, which helps in a device that runs on limited power supply such as a mobile phone powered by a battery.

3DES is an improvement over standard DES, by expanding the key space. And just as in AES, the immunity against brute force attack can be improved by utilizing the rest of

the available bytes in the key. However, due to the smaller key space (24 bytes against 32), AES can offer a better immunity. But comparing both algorithms using a similar key, based on the results, 3DES is more secure than AES.

3DES is useful in applications where memory is a scars resource, and it is also suitable for systems that are based on DES, which can be upgraded to 3DES with minimum modifications.

Blowfish performed very well in all three tests with all different key lengths, it had the largest brute force attack time even when the smallest key length was used. Encryption and decryption times were less than the ones obtained from 3DES, and the brute force attack times where similar for the same key when different text sizes were used.

The flexibility Blowfish offer from the key length size is a very important advantage. Larger keys can be used according to the level of security needed, since based on the results, the immunity against brute force attacks is increased by increasing the key size, so increasing the key length and utilizing the all available bytes in the key can yield to a very robust and secure encryption, without compromising the encryption and decryption times.

Based on the results obtained from the tests on the computer and on the Raspberry Pi, blowfish encryption algorithm is recommended for applications with high security requirements, and the flexible key lengths makes it suitable for applications with limited memory.

Future work

The future work could involve observing the effect different encryption modes have on the encryption and decryption times, such as Cipher-block chaining, and Cipher feedback, as well as the effect they have on the immunity against brute force attacks.

Different encryptions should also be considered such as elliptic curve cryptography, Twofish and Threefish, with larger key space if possible.

Designing a dedicated system for testing the encryption algorithms with different programming language might also be considered, in order to minimize the background application interference.

Different attacks can be considered beside the brute force attacks, focusing on one encryption algorithm at a time.

Using an actual smartphone instead of the Raspberry Pi for testing different encryption algorithms, can yield a more accurate reading on the encryption and decryption times, as well as a precise reading for the power consumption.

References

Atmel (2010). *Considerations for RFID Technology Selection* [online] [cited 12 Feb. 2012]. Available from the Internet <URL: http://www.docstoc.com/docs/44727761/Considerations-for-RFID-Technology-Selection>

Balanis, Constantine A. (2005). *Antenna Theory: Analysis and Design*. John Wiley and Sons, Inc. 1136 p. ISBN: 978-0-471-66782-7.

Beazley, David and Brian K. Jones (2013). *Python Cookbook* 3rd Edition, O'reily, ISBN:978-1-449-34037-7

Burkard, Simon (2013). *Near Field Communication in Smartphones.* Berlin Institute of Technology, Germany Available from the Internet <URL https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/nfc-in-smartphones_burkard.pdf

Cisco (2008). *Wi-Fi Location-Based Services 4.1 Design Guide*.158 p. Available from the Internet <URL: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Mobility/WiFiLBS-DG.pdf

Cardlogix (2009). *Smart Card Security Basics*. Cardlogix publication . Available from the Internet <URL:www.smartcardbasics.com

ETSI (2003). *Near Field Communication (NFC)IP-1 : Interface and Protocol (NFCIP-1).*[online] [cited 8 May. 2012]. Available from the Internet <URL:http://www.etsi.org/deliver/etsi_ts/102100_102199/102190/01.01.01_60/ts_102190v010101p.pdf>

Finkenzeller, Klaus (2010). RFID HANDBOOK 3rd edition. Wiltshire, UK. John Wiley & Sons. ISBN 978-0-470-69506-7

Gatliff, Bill (2003). *Encrypting data with the Blowfish algorithm.* Available from the Internet <URL:http://www.embedded.com/design/configurable-systems/4024599/Encrypting-data-with-the-Blowfish-algorithm

Hioki, Warren (2000). *Telecommunications*. 4th Ed. New Jersey, USA: Prentice Hall. ISBN: 013020031X.

Hossein Motlagh, Naser (2012). *Near Field Communications (NFC) a Technical Overview*. University of Vaasa

Haselsteiner, Ernst & Klemens Breitfuß (2006). *Security in Near Field Communication (NFC)*. [online] [cited 8 May. 2012]. Available from the Internet <URL:http://ece.wpi.edu/~dchasaki/papers/Security%20in%20NFC.pdf

ISO/IEC 7810 (2002). *Identification cards — Physical characteristics*

ISO/IEC 14443-3 (1999). *Identification cards - Contactless integrated circuit(s) cards- Proximity cards - Part 3: Initialization and anti-collision*

ISO/IEC 14443-3 (1998). *Identification cards - Contactless integrated circuit(s) cards- Proximity cards - Part 4: Transmission Protocol*

ISO/IEC 15693-3 (2007). *Identification cards - Contactless integrated circuit(s) cards - Vicinity cards - Part 3: Anti-collision and transmission protocol*

ISO/IEC 18092 (2013). *Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol (NFCIP-1)*

Madlmayr, G. J. Langer, C. Kantner & J Scharinger (2008). *NFC Devices: Security and Privacy* . Univ. of Applied Science of Upper Austria, The Third International Conference on Availability, Reliability and Security, Barcelona Available from the Internet

<URL:http://ieeexplore.ieee.org.proxy.tritonia.fi/xpl/articleDetails.jsp?tp=&arnumber=4529403&queryText%3DNFC+Devices%3A+Security+and+Privacy

Martelli, Alex (2003), *Python in a Nutshell* 2nd edition, O'riely. ISBN:978-0-596-10046-9

Mulliner, Collin (2009). *Vulnerability Analysis and Attacks on NFC-enabled Mobile Phones*. IWSS March 2009 Fukuoka Japan. Available from the Internet <URL:http://www.mulliner.org/nfc/

Rankl , Wolfgang & Wolfgang Effing (2010). *Smart Card Handbook*. 4th Edition, John Wiley & Sons. ISBN: 978-0-470-74367-6

Penttonen, Martti (2011). *Data Security*. University of Eastern Finland

Qian, Yang (2013). *Applied Cryptography in Embedded Systems*. University of Vaasa

Stalling, William (2011). *Cryptography and Network Security*, 5th Edition. Prentice Hall.

Standard ECMA-373 (2012). *Near Field Communication Wired Interface (NFC-WI)*, 2nd Edition.

Ul-Haq, Salman, Jawad Masood, Aamir Majeed & Usman Aziz (2011). *Bulk Encryption on GPUS*. Available from the Internet <URL: http://developer.amd.com/resources/documentation-articles/articles-whitepapers/bulk-encryption-on-gpus/

Van Damme, Gauthier & Karel Wouters (2009). *Practical Experiences with NFC Security on mobile Phones*. Katholieke Universiteit Leuven. Available from the Internet<URL:http://www.cosic.esat.kuleuven.be/publications/article-1288.pdf

www.gsmarena.com/qualcomm_unveils_snapdragon_810_and_808_64bit_chipsets-news-8241.php

Appendices

Appendix 1: Caesar Source Code.

```
########################################
 ##                                    ##
  ##  Caesar by Mohammed A. Mohammed  ##
 ##                                    ##
########################################

import time
import timeit

# this function is for deciding whether the user wants to encrypt or decrypt
def getMode():
    while True:
        print('Do you wish to Encrypt "e", Decrypt "d" or Brute force "b" a message?')
        mode = input()
        if mode in 'e d b'.split():
            return mode
        else:
            print('Enter either "e" for encrypt,"d" for decrypt, or "b" for brute force.')

#this funiction is for the user to input the plain/ciphered text
def text():
    print('Enter text:')
    return input()

#this funiction is for choosing the key
def getKey():
    key = 0
    while True:
        print('Enter the key number')
        key = int(input())
        key = key%95 # the key is set in module 95 so it will cover the range of all ASCII values
without exceeding it
        return key

#this funiction is for the encryption/decryption
def getTranslatedMessage(mode, text, key):
    start = time.clock()
    if mode[0] != 'b':
        if mode[0] == 'd':
            key = -key # since the encryption and decryption is Caesar are the exact opposite, the if
statment only flips the sign of the key
        translated = ''

        for symbol in text:
            num = ord(symbol) # getting the numerical value for the symbols (between 32-126)
            mnum = num-32 # tranfaring the value of the symbol before shifting it to the range of
(0-94)
```

```
            msnum = (mnum+key)%95 # shifting the modified value of the key within the allowed
range (0-94)
            snum = msnum+32 # returning the value to the ASCII range (32-126)
            translated += chr(snum) # adding the new value to the new sring
    end = time.clock()
    print ("execution time is", end-start)
    return translated

mode = getMode()
text = text()
if mode[0] != "b":
    key = getKey()
    print('Your translated text is:')
    print(getTranslatedMessage(mode, text, key))
else:
    print("enter a part of the plain text")
    text2 = input("")
    for key in range (0, 95, 1):
        start = time.clock()
        translated = ("")
        for symbol in text:
            num = ord(symbol)
            mnum = num-32
            msnum = (mnum+key)%95
            snum = msnum+32
            translated += chr(snum)

        if translated.find(text2) != -1:
            print ("\nthe key is =",key)
            print ("\nand the palin text is:\n")
            print (translated)
            end = time.clock()
            print ("execution time is", end-start)
```

Appendix 2: DES Source Code.

```
#######################################
 ##                                   ##
  ##  DES by Mohammed A. Mohammed  ##
 ##                                   ##
#######################################

import time
import timeit
from Crypto.Cipher import DES #encryption is used for disguising data
import os # os is for urandom, which is an accepted producer of randomness that is suitable for cryptology.
import base64

print ("key size is fixed at 8 Bytes including parities (56 bits effective)")
size = 8

key = '99999999'
cipher = DES.new(key)
BS = 8
ciphertext = ""
testtext = ""

# deciding to encrypt, decrypt, or brute force a text.
def getMode():
    while True:
        mode = input('Do you wish to Encrypt "e", Decrypt "d" or Brute force "b" a message?\n')
        if mode in 'e d b'.split():
            return mode
        else:
            print('Enter either "e" for encrypt,"d" for decrypt, or "b" for brute force.\n')

# encryption function
def encrypt(text):
    start = time.clock()
    length = len(plaintext)
    pad = lambda s: s + (BS - len(s) % BS) * ('~')
    paddedtext = pad(plaintext)
    encrypted = DES.new(key, DES.MODE_ECB)
    ciphertext = base64.b64encode(encrypted.encrypt(paddedtext)).decode("utf-8")
    end = time.clock()
    print ("execution time is", end-start)
    return ciphertext

# decryption function
def decrypt(text):
    start = time.clock()
    decrypted = DES.new(key, DES.MODE_ECB)
    paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("utf-8")
    l = paddedtext.count ('~')
    end = time.clock()
```

```
    print ("execution time is", end-start)
    return paddedtext[:len(paddedtext)-l]

#  brute force attack function
# first part, generating the test keys
def testkeys ():
    start = time.clock()
    for i in range (100000000):
        temp = '{:08d}'.format (i)
        if (brute(temp)) == 1:
            end = time.clock()
            print ("execution time is", end-start)
            break

# second part, checking the test keys to decrypt the cipher text
def brute(testkey):
        testkey = format (testkey)
        decrypted = DES.new(testkey, DES.MODE_ECB)
        paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("latin-1")

        if paddedtext.find(testtext) != -1:
            print ("the key is ",testkey)
            l = paddedtext.count ('~')
            print (paddedtext[:len(paddedtext)-l])
            return 1
        else:
            return 0

# the begining of the program
mode = getMode()

print ("key is:", key)

if mode[0] == 'e':
    plaintext = input("Enter the plaintext: ")
    encrypted = encrypt(plaintext)
    print ("encrypted:", encrypted)

elif mode [0] == 'd':
    cipher = input("Enter the ciphertext: ")
    decrypted = decrypt(cipher)
    print ("decrypted:\n", decrypted)

else:
    cipher = input("Enter the ciphertext: ")
    testtext = input("enter a part of the plain text: ")
    testkeys ()
```

Appendix 3: 3DES Source Code.

```
########################################
 ##                                    ##
  ##  3DES by Mohammed A. Mohammed  ##
 ##                                    ##
########################################

import time
import timeit
from Crypto.Cipher import DES3 #encryption is used for disguising data
import os # os is for urandom, which is an accepted producer of randomness that is suitable for cryptology.
import base64

print ("choose key size")
size = int(input())
while not (size in [16,24]):
    print("wrong key size")
    print ("choose a suitable key size")
    size = int(input())

key = '0000000099999999'
cipher = DES3.new(key)
BS = 8
ciphertext = ""
testtext = ""

# deciding to encrypt, decrypt, or brute force a text.
def getMode():
    while True:
        mode = input('Do you wish to Encrypt "e", Decrypt "d" or Brute force "b" a message?\n')
        if mode in 'e d b'.split():
            return mode
        else:
            print('Enter either "e" for encrypt,"d" for decrypt, or "b" for brute force.\n')

# encryption function
def encrypt(text):
    start = time.clock()
    length = len(plaintext)
    pad = lambda s: s + (BS - len(s) % BS) * ('~')
    paddedtext = pad(plaintext)
    encrypted = DES3.new(key, DES3.MODE_ECB)
    ciphertext = base64.b64encode(encrypted.encrypt(paddedtext)).decode("utf-8")
    end = time.clock()
    print ("execution time is", end-start)
    return ciphertext

# decryption function
def decrypt(text):
```

```
    start = time.clock()
    decrypted = DES3.new(key, DES3.MODE_ECB)
    paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("utf-8")
    l = paddedtext.count ('~')
    end = time.clock()
    print ("execution time is", end-start)
    return paddedtext[:len(paddedtext)-l]

#  brute force attack function
# first part, generating the test keys
def testkeys ():
    start = time.clock()
    for i in range (10000000000000000):
        temp = '{:016d}'.format (i)
        if (brute(temp)) == 1:
            end = time.clock()
            print ("execution time is", end-start)
            break

# second part, checking the test keys to decrypt the cipher text
def brute(testkey):
        testkey = format (testkey)
        decrypted = DES3.new(testkey, DES3.MODE_ECB)
        paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("latin-1")

        if paddedtext.find(testtext) != -1:
            print ("the key is ",testkey)
            l = paddedtext.count ('~')
            print (paddedtext[:len(paddedtext)-l])
            return 1
        else:
            return 0

# the begining of the program
mode = getMode()

print ("key is:", key)

if mode[0] == 'e':
    plaintext = input("Enter the plaintext: ")
    encrypted = encrypt(plaintext)
    print ("encrypted:", encrypted)

elif mode [0] == 'd':
    cipher = input("Enter the ciphertext: ")
    decrypted = decrypt(cipher)
    print ("decrypted:\n", decrypted)

else:
    cipher = input("Enter the ciphertext: ")
    testtext = input("enter a part of the plain text: ")
    testkeys ()
```

Appendix 4: AES Source Code.

```
#######################################
 ##                                   ##
  ##  AES by Mohammed A. Mohammed  ##
 ##                                   ##
#######################################

import time
import timeit
from Crypto.Cipher import AES #encryption is used for disguising data
import os # os is for urandom, which is an accepted producer of randomness that is suitable for cryptology.
import base64

print ("choose key size")
size = int(input())
while not (size in [16,24,32]):
    print("wrong key size")
    print ("choose a suitable key size")
    size = int(input())

key = '0000000099999999'
cipher = AES.new(key)
BS = 16
ciphertext = ""
testtext = ""

# deciding to encrypt, decrypt, or brute force a text.
def getMode():
    while True:
        mode = input('Do you wish to Encrypt "e", Decrypt "d" or Brute force "b" a message?\n')
        if mode in 'e d b'.split():
            return mode
        else:
            print('Enter either "e" for encrypt,"d" for decrypt, or "b" for brute force.\n')

# encryption function
def encrypt(text):
    start = time.clock()
    length = len(plaintext)
    pad = lambda s: s + (BS - len(s) % BS) * ('~')
    paddedtext = pad(plaintext)
    encrypted = AES.new(key, AES.MODE_ECB)
    ciphertext = base64.b64encode(encrypted.encrypt(paddedtext)).decode("utf-8")
    end = time.clock()
    print ("execution time is", end-start)
    return ciphertext

# decryption function
def decrypt(text):
    start = time.clock()
```

```
    decrypted = AES.new(key, AES.MODE_ECB)
    paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("utf-8")
    l = paddedtext.count ('~')
    end = time.clock()
    print ("execution time is", end-start)
    return paddedtext[:len(paddedtext)-l]

#  brute force attack function
# first part, generating the test keys
def testkeys ():
    start = time.clock()
    for i in range (10000000000000000):
        temp = '{:016d}'.format (i)
        if (brute(temp)) == 1:
            end = time.clock()
            print ("execution time is", end-start)
            break

# second part, checking the test keys to decrypt the cipher text
def brute(testkey):
        testkey = format (testkey)
        decrypted = AES.new(testkey, AES.MODE_ECB)
        paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("latin-1")

        if paddedtext.find(testtext) != -1:
            print ("the key is ",testkey)
            l = paddedtext.count ('~')
            print (paddedtext[:len(paddedtext)-l])
            return 1
        else:
            return 0

# the begining of the program
mode = getMode()

print ("key is:", key)

if mode[0] == 'e':
    plaintext = input("Enter the plaintext: ")
    encrypted = encrypt(plaintext)
    print ("encrypted:", encrypted)

elif mode [0] == 'd':
    cipher = input("Enter the ciphertext: ")
    decrypted = decrypt(cipher)
    print ("decrypted:\n", decrypted)

else:
    cipher = input("Enter the ciphertext: ")
    testtext = input("enter a part of the plain text: ")
    testkeys ()
```

Appendix 5: Blowfish Source Code.

```
##########################################
 ##                                      ##
  ##  Blowfish by Mohammed A. Mohammed  ##
 ##                                      ##
##########################################

import time
import timeit
from Crypto.Cipher import  #encryption is used for disguising data
import os # os is for urandom, which is an accepted producer of randomness that is suitable for cryptology.
import base64

print ("choose key size")
size = int(input())
while not (size in range (4, 57)):
    print("wrong key size")
    print ("choose a suitable key size")
    size = int(input())

key = '99999999'
cipher = Blowfish.new(key)
BS = 8
ciphertext = ""
testtext = ""

# deciding to encrypt, decrypt, or brute force a text.
def getMode():
    while True:
        mode = input('Do you wish to Encrypt "e", Decrypt "d" or Brute force "b" a message?\n')
        if mode in 'e d b'.split():
            return mode
        else:
            print('Enter either "e" for encrypt,"d" for decrypt, or "b" for brute force.\n')

# encryption function
def encrypt(text):
    start = time.clock()
    length = len(plaintext)
    pad = lambda s: s + (BS - len(s) % BS) * ('~')
    paddedtext = pad(plaintext)
    encrypted = Blowfish.new(key, Blowfish.MODE_ECB)
    ciphertext = base64.b64encode(encrypted.encrypt(paddedtext)).decode("utf-8")
    end = time.clock()
    print ("execution time is", end-start)
    return ciphertext

# decryption function
def decrypt(text):
    start = time.clock()
```

```
    decrypted = Blowfish.new(key, Blowfish.MODE_ECB)
    paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("utf-8")
    l = paddedtext.count ('~')
    end = time.clock()
    print ("execution time is", end-start)
    return paddedtext[:len(paddedtext)-l]

#  brute force attack function
# first part, generating the test keys
def testkeys ():
    start = time.clock()
    for i in range (10000000000000000):
        temp = '{:08d}'.format (i)
        if (brute(temp)) == 1:
            end = time.clock()
            print ("execution time is", end-start)
            break

# second part, checking the test keys to decrypt the cipher text
def brute(testkey):
        testkey = format (testkey)
        decrypted = Blowfish.new(testkey, Blowfish.MODE_ECB)
        paddedtext = decrypted.decrypt(base64.b64decode(cipher)).decode("latin-1")

        if paddedtext.find(testtext) != -1:
            print ("the key is ",testkey)
            l = paddedtext.count ('~')
            print (paddedtext[:len(paddedtext)-l])
            return 1
        else:
            return 0

# the begining of the program
mode = getMode()

print ("key is:", key)

if mode[0] == 'e':
    plaintext = input("Enter the plaintext: ")
    encrypted = encrypt(plaintext)
    print ("encrypted:", encrypted)

elif mode [0] == 'd':
    cipher = input("Enter the ciphertext: ")
    decrypted = decrypt(cipher)
    print ("decrypted:\n", decrypted)

else:
    cipher = input("Enter the ciphertext: ")
    testtext = input("enter a part of the plain text: ")
    testkeys ()
```