

# ADD ON SECURITY MODULE FOR WIRELESS SETS



By

Hafiza Fareeha Jabeen

Imama Ghazanfar

Shafique Ur Rehman

Submitted to the Faculty of Department of Electrical Engineering,  
Military College of Signals, National University of Science and Technology, Islamabad

In partial fulfilment for the requirements of a B.E Degree in

Telecom Engineering

JUNE 2016

## ABSTRACT

### ADD ON SECURITY MODULE FOR WIRELESS SETS

In today's high technology environment, organizations like military are becoming more and more dependent on their information systems. Information security is one of the most important military issues of the 21st century. Thus, we need to reduce the risk of data breach by developing a low cost and trust worthy module which could not only be locally produced but shall also be compatible with military communication devices like wireless sets. The project focuses on development of a minimum cost module with implementation of the advanced encryption standard (AES) algorithm and serial communication on Raspberry pi board. The module thus produced shall be used with military wireless sets via RS232 communication standard.

DEDICATED TO

Allah Almighty,

Our parents, friends and family for their support and prayers,

Our supervisor,

And all faculty members for their help.

## ACKNOWLEDGEMENTS

First and foremost, all praises to the Almighty Allah for giving us strength and continuous shower of His blessings.

We would also like to thank our project supervisor Lt. Col. Saifullah Khalid for showing trust in our capabilities and giving us the opportunity to accomplish this task. His motivation and help deeply inspired us.

We would also like to thank all the respectable faculty members for always helping us throughout this period and for their golden advices.

Special gratitude to our parents and families who continuously supported us through their prayers and encouragement.

Lastly, we would like to thank our colleagues who always helped us to the best of their ability and our dear friends for not only helping us and for also building our morale.

## TABLE OF CONTENTS

1. <u>Introduction</u>	<u>1</u>
1.1 <u>Background and Motivation</u>	<u>1</u>
1.2 <u>Project Description and Salient Features</u>	<u>1</u>
1.3 <u>Scope</u>	<u>2</u>
1.4 <u>Objectives</u>	<u>2</u>
1.4.1 <u>Secured Data Transmission</u>	<u>2</u>
1.4.2 <u>Low Cost Module</u>	<u>2</u>
1.5 <u>Specifications</u>	<u>3</u>
1.6 <u>Deliverables</u>	<u>3</u>
2. <u>Literature Review</u>	<u>5</u>
2.1 <u>Serial Port Communication</u>	<u>5</u>
2.1.1 <u>RS232 Standard</u>	<u>5</u>
2.1.2 <u>Serial Port: DTE (PC) and DCE (Modem)</u>	<u>6</u>
2.1.3 <u>Hardware Flow Control</u>	<u>7</u>
2.2 <u>Advance Encryption Standard</u>	<u>7</u>
2.2.1 <u>AES methodology</u>	<u>7</u>
2.3 <u>Secure Hash Algorithm</u>	<u>9</u>
3. <u>Design and Development</u>	<u>11</u>
3.1 <u>Hardware</u>	<u>11</u>
3.1.1 <u>RS-232 Connectors and Cables</u>	<u>11</u>
3.1.2 <u>Raspberry Pi Board</u>	<u>12</u>

3.1.3	<u>RF-5800h MP</u>	<u>14</u>
3.1.4	<u>TTL to RS232 Converter</u>	<u>15</u>
3.2	<u>Software Implementation</u>	<u>16</u>
3.2.1	<u>Serial communication implementation</u>	<u>16</u>
3.2.2	<u>AES Implementation</u>	<u>17</u>
3.3	<u>Detailed design</u>	<u>17</u>
3.4	<u>Tasks and Challenges</u>	<u>17</u>
4.	<u>Project Analysis and Evaluation</u>	<u>19</u>
4.1	<u>AES Implementation</u>	<u>19</u>
4.2	<u>Serial Communication</u>	<u>20</u>
4.3	<u>Combining AES and Serial Communication</u>	<u>20</u>
4.4	<u>Final Testing</u>	<u>20</u>
4.5	<u>Data Flow Model</u>	<u>21</u>
5.	<u>Recommendations and Conclusion</u>	<u>23</u>
5.1	<u>Overview</u>	<u>23</u>
5.2	<u>Limitations</u>	<u>23</u>
5.3	<u>Recommendations</u>	<u>23</u>
5.4	<u>Conclusion</u>	<u>23</u>
6.	<u>Bibliography</u>	<u>26</u>
6.1	<u>General References</u>	<u>26</u>
6.2	<u>Online help</u>	<u>26</u>
6.3	<u>In Text Citations</u>	<u>26</u>
7.	<u>Appendix-A: Codes</u>	<u>29</u>

## LIST OF FIGURES

<u>Figure 1-1: Crypto Ag HC-2605 terminal</u>	<u>3</u>
<u>Figure 2-1: Connection between two devices communicating Through Rs232 standard</u>	<u>5</u>
<u>Figure 2-2: Male and female connectors (Pin configuration)</u>	<u>6</u>
<u>Figure 2-3: NULL Modem</u>	<u>7</u>
<u>Figure 2-4: Structure of Key and Input Data</u>	<u>8</u>
<u>Figure 2-5: Subbytes operation</u>	<u>8</u>
<u>Figure 2-6: Shiftrows operation</u>	<u>8</u>
<u>Figure 2-7: MixColumn operation</u>	<u>9</u>
<u>Figure 2-8: Add round key</u>	<u>9</u>
<u>Figure 3-1: RS232 Connectors</u>	<u>12</u>
<u>Figure 3-2: Features of Raspberry Pi</u>	<u>13</u>
<u>Figure 3-3: RF 5800-h MP</u>	<u>15</u>
<u>Figure 3-4: Timing diagram of TTL and RS232</u>	<u>16</u>
<u>Figure 3-5: TTL to RS232 logic converter</u>	<u>16</u>
<u>Figure 3-6: Detailed Design</u>	<u>17</u>
<u>Figure 4-1: Finalized Design</u>	<u>19</u>

Figure 4-2: Data Flow Model (sending end) 21

Figure 4-3: Data Flow Model (receiving end) 22



## LIST OF TABLES

<u>Table 2.1: Pin configuration of 9 DB9 female/male connector</u>	<u>6</u>
<u>Table 3-1: Raspberry pi 1 model B specs</u>	<u>14</u>
<u>Table 3-2: Raspberry pi 1 model B+ specs</u>	<u>14</u>

## LIST OF ABBREVIATIONS

AES	Advance encryption standard
DB	D-shell Body
DCE	Data Circuit-Terminating Equipment
DES	Data Encryption Standard
DTE	Data Terminal Equipment
HF	High Frequency
NIST	National Institute of Standards and Technology
OS	Operating System
PC	Personal Computer
RF	Radio Frequency
RS	Recommended Standard

# **1. Introduction**

The project focuses on the design of an ADD-ON security module for wireless radio sets which are being widely used by soldiers in battle field. The project has been designed to provide conservancy of confidentiality of data sent over wireless sets. It focuses on design and development of a security module which can be connected externally to wireless sets and can provide encryption of all the data before sending it to wireless set and also decryption of the encrypted data externally after receiving it from wireless set and security module.

## **1.1 Background and motivation**

The Add-on module provides highly secure data communication over high frequency radio sets. The module uses the art of encryption and serial communication techniques. Currently available encryption in HF radio set used in military is of foreign origin, has limited security value and is unable to fetch user confidence. Therefore, designing an external add-on security module would not only provide better security but would also satisfy user up to greater extent. The module is used along with in effect radio sets for end-to-end encryption without the compulsion of varying radio communication infrastructures.

Encryption/Decryption is used to defend sensitive information from unauthorized access, use, disclosure, modification or destruction. It is a general term that can be used regardless of the form the data. The field is growing rapidly as the need for securing the data with the growth in the technology is becoming necessity.

## **1.2 Project Description and salient features**

The project is intended to provide a low cost reliable add-on security module for wireless radio sets. It will serve two main purposes of securing the data and providing cost efficiency. A locally developed low cost, trustworthy Security Module is a need of the soldiers in the era where Data breaching and intelligence failure are considered as worst in the fate of the nations. The module will help army men to have better as well as reliable communication with complete trust and without having a fear of the information stealing. The module will also be a cheaper one than those currently available as it will be produced

locally. The project will provide a platform for secure tactical chat between soldiers in battle field with minimum errors and delay time.

### **1.3 Scope**

The project basically involves the design of a hardware architecture, for people longing for to achieve confidential data communication like army men especially those deployed in battlefield, that will not only ensure the security of all the data passing through the wireless sets (extensively used in battle field), but will also aim at overcoming the cost inefficiency of security modules currently available in the market. The scope of the project includes Secure and reliable wireless communication using some communication technique compatible with hf radio sets. Preserving the confidentiality of data that needs to be transmitted by a user, precisely a soldier, over wireless radio set. The project also serves national goal as it is specifically made for army radio sets which are imported from other countries and have foreign origin based encryption techniques employed in them.

### **1.4 Objectives**

The project served the main objectives of securing the data transmission over wireless sets and reducing cost inefficiency of currently available modules.

#### **1.4.1 Secured Data Transmission**

RF-5800h-mp used for testing purpose in our project is a member of the falcon ii® family of multiband tactical radio systems. It provides communications of voice and data. The radio set has its own encryption algorithm Citadel which is embedded in Citadel ASIC. But the problem is that the algorithm as well as hardware is of foreign origin. Foreign equipment cannot be trusted due to presence of backdoors especially when it comes to areas as sensitive as battlefield communication. Thus we need an encryption module which shall be locally developed and whose key is only known to soldier. Thus a locally developed encryption module was designed.

#### **1.4.2 Low cost module**

The other main objective was to provide a low cost solution for the problem stated above. There are some modules currently available in market like Crypto Ag HC-2605 terminal which can be used as an additional security module but its cost is very high ranging from

PKR-20000 to PKR-35000 per module and number of required modules is as large as 25000. A module with such high cost cannot be provided for each and every wireless set thereby making it an ineffective module in terms of affordability which is a great factor in deciding the success of a product in rapport of fetching the trust of user and making customers. The module thus developed is a low cost one and was developed using the currently available resources.



**Figure 1-1:** Crypto Ag HC-2605 terminal

## **1.5 Specifications**

The hardware of the project has been developed using an embedded Raspberry bi board 1 class B and B+.

The module has been programmed by c, c++ which is the highly hardware descriptive language.

The technique used for encryption and decryption is Advanced Encryption Standard-256 (AES-256).

SHA-1 algorithm developed in c++ is used to maintain security of key and passwords used for encryption and decryption.

After development of module, the module is also made compatible to radios by converting its TTL logic for serial communication to RS232 logic.

## **1.6 Deliverables**

The project was aimed to deliver a working add-on security module for reliable and secure data communication using RF-5800H. At the transmitting end the module sends cipher text

of data provided by user serially to radio set and at the receiving end, it serially receives data from radio set, decrypts it and display is on screen.

## 2. Literature Review

Our project has two main components. One is Serial communication using RS232 standard and other is AES 128 encryption/decryption. This chapter will give background of both.

### 2.1 Serial Port Communication

Serial communication implies sending data bit by bit over a single wire. Data rate for the link must be the same for the transmitter and the receiver. <sup>[1]</sup>RS-232 is a standard for serial communication transmission of data. This section of document explains about RS232 connection working between rpi and other computer or radio.

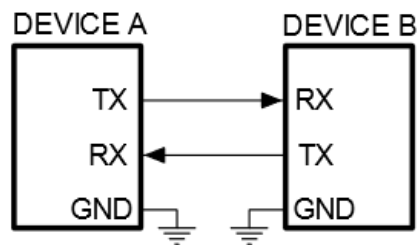


Figure 2-1: Connection between two devices communicating through Rs232 standard.

The serial port on radio used for testing purpose is a half-duplex i-e it can either send or receive data at the same time and uses same communication channel for transmission as well as reception of data. Thus, it used RTS/CTS handshake protocol.

#### 2.1.1 RS232 Standard

RS stands for “Recommended Standard”. The standard was established by a committee of standards now known as Electronic Industries Association in 1960s. It defines the mechanical and electrical characteristics of the connection including handshake pins and the function of the signals, the voltage levels and maximum bit rate. The standard also defines how computers (DTEs) connect to modems (DCEs). Mainly two configurations are characteristically used: One for a 9-pin connector and the other for a 25-pin connector. We used 9 pin configuration for radio.

### 2.1.2 Serial Port: DTE (PC) and DCE (Modem)

DTE stands for Data Terminal Equipment and DCE stands for Data Communications Equipment. Both these terms are used to specify the direction of the signals on the pins and the pin-out for the connectors on a device. The system we are using (Raspberry Pi) is a DTE device, while most other devices (radio) are usually DCE devices.

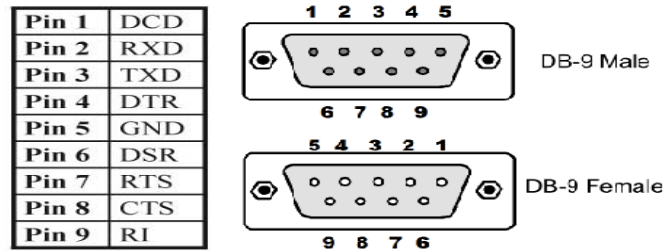


Figure 2-2: Male and female connectors (Pin configuration)

Pin #	Direction of signal
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from a modem

Table 2-1: Pin configuration of 9 DB9 female/male connector.



DTE devices usually use a 25-pin male connector, and DCE devices use a 25-pin female connector. In order to connect two DTE devices, NULL Modem is used.

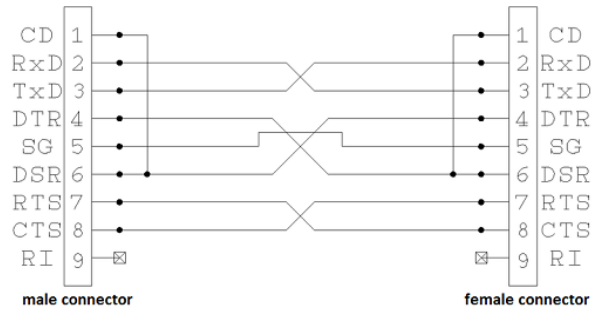


Figure 2-3: NULL Modem.

### 2.1.3 Hardware Flow Control

RTS stands for Request To Send and CTS stands for Clear To Send. RTS and CTS are used when "hardware flow control" is enabled in both the DTE and DCE devices. The DTE puts RTS high when it is ready and able to transmit data. The DCE device puts CTS in high condition to tell the DTE device that it is ready to receive the data. If the DCE is unable to receive data puts CTS in low condition. Together, these two lines make up "hardware flow control".

## 2.2 Advance Encryption Standard

In January 1997 the National Institute of Standards and Technology (NIST) initiated the search for a replacement for the Data Encryption Standard (DES). The requirements for new standards were that it should be an 18 bit block cipher with the choice of three key sizes i.e. 128,192,256 bits, it should be a public design and it should be secure and available royalty-free worldwide. At the conclusion of this standardization effort, with many man-years of cryptanalytic and implementation expertise provided from around the world, Rijndael, developed by John Daemen and Vincent Rijmen was selected as AES. <sup>[2]</sup>

### 2.2.1 AES methodology

The AES is a classic substitution/permutation network that requires 10, 12 or 14 rounds of encryption. The exact number depends upon the length of the key. Using the nomenclature of FIPS 197, a typical round of the cipher uses the four operations namely substitute bytes,

shift rows, mix columns and add round key. Before encryption/decryption both the key and the input data (state) are structured in a 4x4 matrix of bytes.<sup>[3]</sup>

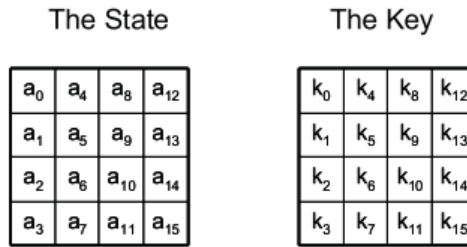


Figure 2-4: Structure of Key and Input Data

Substitute Bytes Operation:

The Subbytes operation is a nonlinear substitution. It can be interpreted in different ways. It can be considered as a lookup table. With the help of this lookup table, each of the 16 bytes of the state is substituted by the corresponding values found in the table

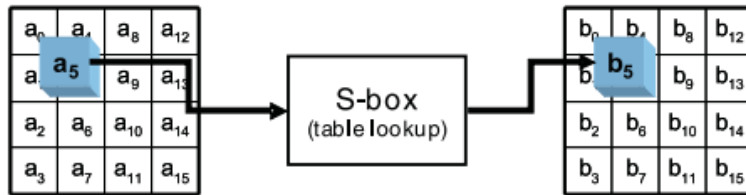


Figure 2-5: Subbytes operation

Shift Rows Operation:

In this operation different rows of the 4x4 input data are processed. The first row remains unchanged. The second row is shifted one byte to the left in the matrix, the third row is shifted two bytes, and the fourth row is shifted three bytes.

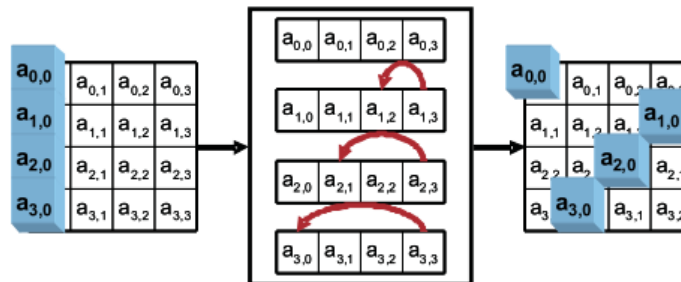


Figure 2-6: Shiftrows operation

### Mix Columns Operation:

This is the most complex operation. This operation is performed in two steps. First is Matrix Multiplication and second one is Galois Field. In Matrix multiplication, the multiplication is performed one column at a time. Each value in the column is multiplied against every value of the matrix (16 total multiplications). The results of these multiplications are XOR'ed together to produce only 4 result bytes for the next state. Therefore 4 bytes input, 16 multiplications 12 XORs and 4 bytes output. The multiplication is performed one matrix row at a time against each value of a state column. Whereas in Galois Field Multiplication, the multiplication is performed over a Galois Field with the help of two tables called E table and L table. The result of the multiplication is simply the result of a lookup of the L table, followed by the addition of the results, followed by a lookup up to the E table. The addition is a regular mathematical addition represented by +, not a bitwise AND.<sup>[4]</sup>

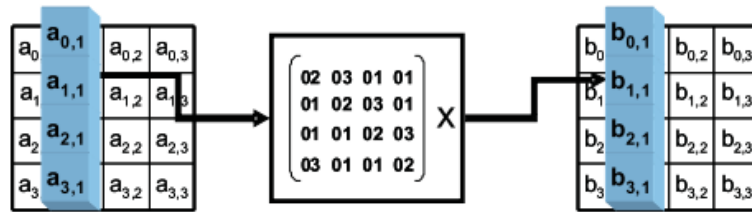


Figure 2-7: MixColumn operation.

### Add Round Key:

The corresponding bytes of the input data and the expanded key are XOR'ed in this step

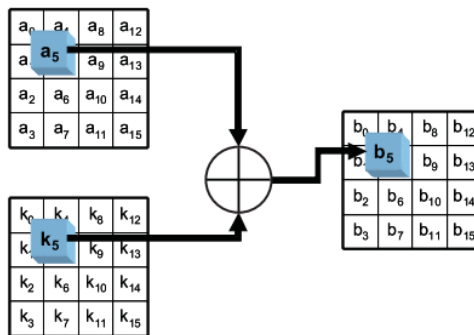


Figure 2-8: Add round key

## 2.3 Secure Hash Algorithm (SHA)

SHA is a cryptographic hash function. It was designed by US NSA and is a US FIPS standard published by NIST US. Three successive SHA algorithms were developed named: SHA-1, SHA-2 and SHA-3. The original specification of the algorithm was published in 1993 by U.S. SHA is widely used in security application and protocols. The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

When a message of any length less than  $2^{64}$  bits is input, a 160-bit output is produced by SHA-1 which is called message digest. The message digest can input to a signature algorithm which generates or verifies the signature for the message. The same hash algorithm must be used by both the verifier and creator of the digital signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature. In this algorithm, following logical operations are applied on words.<sup>[5]</sup>

#### **Message Digest:**

The purpose of message padding is to make the total length of a padded message a multiple of 512. SHA-1 sequentially processes blocks of 512 bits when computing the message digest.<sup>[6]</sup>

All these techniques were thoroughly studied and different methods of their implementations as well as vulnerability of each implementation of encryption techniques were studied. And after all the research the best possible solution was selected and implemented in the project.

### **3. Design and Development**

The aim of our project is to create a cheap security module for wireless radio sets. The module shall be stand alone and shall also be capable of storing and processing all the data and codes. The project can be divided into a hardware and a software part.

The hardware part includes the boards used for implementing the codes and security algorithm with specified memory and processing power. For this purpose we used Raspberry Pi 1 model B and B+ since these models met all our requirements and still were not too costly

For the final testing of the project designed and developed, we used RF5800 h MP radio set which is a high frequency radio set widely used in Pakistan army.

To create compatibility between the modules designed and the radio sets TTL to RS232 logic converters were used. Since the raspberry pi board's gpio pins used for transmission and reception work on TTL logic whereas radio uses RS232 logic for serial communication.

A null modem was also developed between modules and radios to cross connect the transmitters and receivers of both and to also make use of hardware flow control i.e. RTS/CTS handshake which was required to create compatibility with radio set since it uses RTS/CTS handshaking protocol. For the creation of null modem RS232 connectors and cables were used along with female connectors and cables.

The Software part of the project included the development of code for serial communication and AES along with SHA implementation for key protection and security.

The codes were developed in c and c++ languages with wiringPi library used for serial communication. G++ and gcc compilers were used to compile codes in raspberry pi boards.

#### **3.1 Hardware**

##### **3.1.1 RS-232 Connectors and Cables**

The RS232 standard defines at each device which wires will be sending and receiving each of the signal. According to the standard, male connectors have DTE pin functions, and female connectors have DCE pin functions. The standard recommends the D-subminiature 25-pin connector, but does not make it mandatory. Most devices only implement or use a

few of the twenty signals specified in the standard, so connectors and cables with fewer pins are sufficient for most connections, more compact, and less expensive. The standard does not define a maximum cable length, but instead defines the maximum capacitance that a compliant drive circuit must tolerate. A widely used rule of thumb indicates that cables more than 15 m (50 ft) long will have too much capacitance, unless special cables are used. By using low-capacitance cables, full speed communication can be maintained over larger distances up to about 300 m (1,000 ft). For longer distances, other signal standards are better suited to maintain high speed.<sup>[7]</sup>









Type	Port Image	Connector Image
25-Pin Female		
25-Pin Male		
9-Pin Female		
9-Pin Male		

Figure 3-1: RS232 Connectors

### 3.1.2 Raspberry Pi Board

Raspberry pi boards are tiny computers with memory and processing power which can be used in projects to create portable devices. Several generations of Raspberry pi boards have been released so far. All models include on chip operating power and systems. The Raspberry Pi hardware has evolved through several versions that feature variations in memory capacity and peripheral-device support.<sup>[8]</sup>

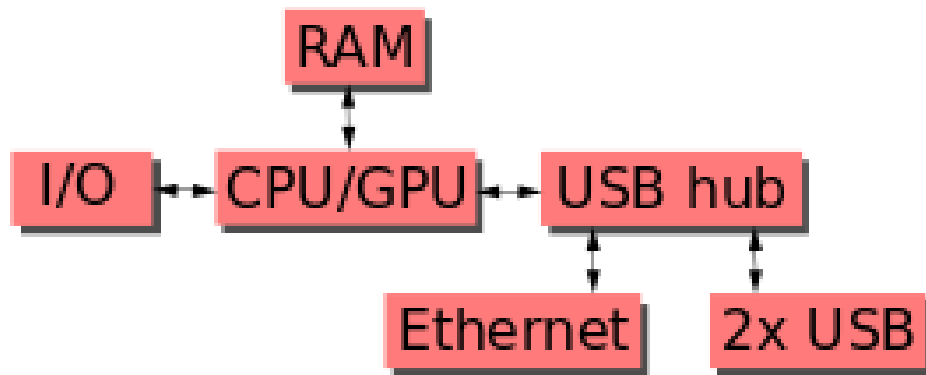


Figure 3-2: Features of Raspberry Pi

All models feature a Broadcom system on a chip (SOC), which includes an ARM compatible CPU and an on chip graphics processing unit GPU. CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM. Secure Digital SD cards are used to store the operating system and program memory in either the SDHC or MicroSDHC sizes. Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phono jack for audio. Lower level output is provided by a number of GPIO pins which support common protocols like I2C. Some models have an 8P8C Ethernet port and the Pi 3 has on board Wi-Fi 802.11n and Bluetooth. The power consumed by the processes required to be run on the raspberry pi board and the memory required decides which board is the best suited for the task. Since in our case we did not require much power and memory so we decided to use the boards with comparatively less memory available to reduce the overall cost of the project and modules

The Raspberry Pi 1 Model B is the first generation Raspberry Pi. It replaced the original Raspberry Pi 1 A in February 2014. Model B is the higher-spec variant of Raspberry Pi 1, with 512 MB of RAM, two USB ports and a 100mb Ethernet port.

On the other hand the model B+ is the newer version of model B with improved memory and more ports. The processing power is also more than model B. The Model B+ is the final modification of the original Raspberry Pi. It replaced the Model B in July 2014 and was superseded by the Raspberry Pi 2 Model B in February 2015. Compared to the Model B it has many specifications improved.

Both boards were used in the project as module 1 and 2.

<ul style="list-style-type: none"> <li>• <b>A 900MHz quad-core ARM Cortex-A7 CPU</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>512MB RAM</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>2 USB ports and 40 GPIO pins</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Full HDMI port</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Ethernet port</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Combined 3.5mm audio jack and composite video</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Micro SD card slot</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>CSI and DSI interface</b></li> </ul>

Table 3-1: Raspberry pi 1 model B specs

<ul style="list-style-type: none"> <li>• <b>A 1800MHz quad-core ARM Cortex-A7 CPU</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>1GB RAM</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>4 USB ports and 40 GPIO pins</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Full HDMI port</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Ethernet port</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Combined 3.5mm audio jack and composite video</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>Micro SD card slot</b></li> </ul>
<ul style="list-style-type: none"> <li>• <b>CSI and DSI interface</b></li> </ul>

Table 3-2: Raspberry pi 1 model B+ specs

### 3.1.3 RF-5800h MP



Designed to provide soldiers with secure voice and data communications, even in the harshest conditions, the RF-5800H-MP provides continuous coverage in the 1.6 to 60 MHz frequency band and enables them to stay connected to mission critical information during operations where line of sight communications are not an option.

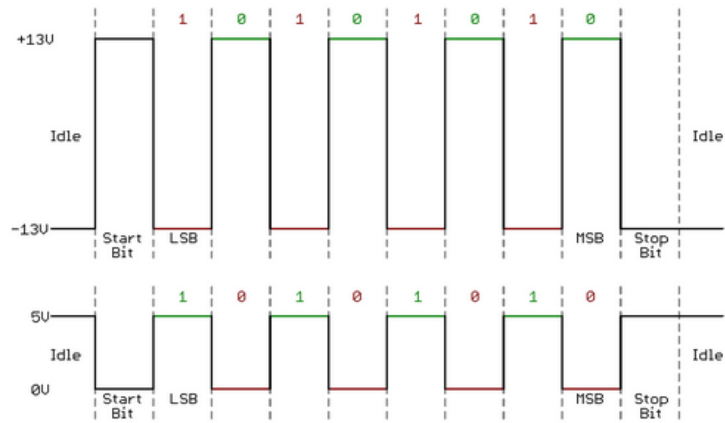


Figure 3-3: RF 5800-h mp

For the final testing of project, data port of RF5800h MP was used.

### 3.1.4 TTL to RS232 Converter

Serial communication at a TTL level will always stay between the limits of 0V and  $V_{cc}$ , which is frequently 5V or 3.3V. A logic high ('1') is signified by  $V_{cc}$ , while a logic low ('0') is 0V. By the RS-232 standard a logic high ('1') is denoted by a negative voltage – anywhere from -3 to -25V – while a logic low ('0') communicates a positive voltage that can be anywhere from +3 to +25V. On most PCs these signals swipec from -13 to +13V. The more extreme voltages of an RS-232 signal help to make it less susceptible to noise, interference, and degradation. This means that an RS-232 signal can largely travel longer physical distances than their TTL counterparts, while still providing a consistent data transmission. The radios use RS232 standard whereas Raspberry pi use TTL logic for this purpose we used a TTL to RS232 converter between both. <sup>[9]</sup>



This timing diagram shows both a TTL (bottom) and RS-232 signal sending 0b01010101

Figure 3-4: Timing diagram of TTL and RS232

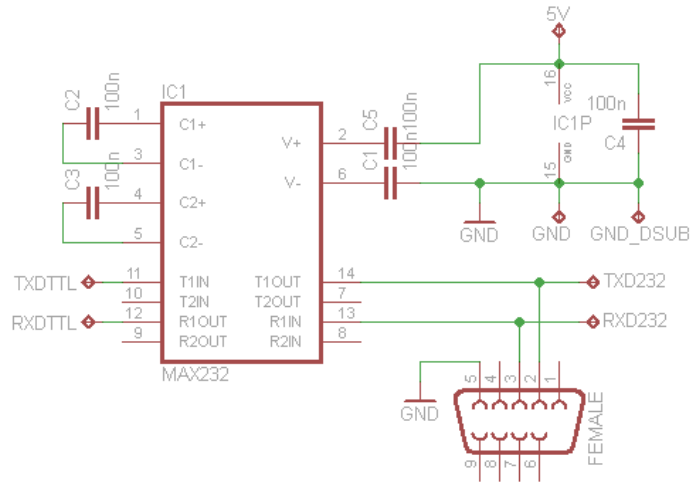


Figure 3-5: TTL to RS232 logic converter

## 3.2 Software Implementation

### 3.2.1 Serial communication implementation

In the first step of our project, serial communication between two PCs was carried out in windows. Hardware Flow control was also used i.e. RTS/CTS handshake protocol was used. For this purpose a code in visual studio using c language using libserial library was developed and then communication was achieved using a null modem configuration between two PCs as shown earlier. The next step in implementation of serial communication was to establish it in Linux environment and then import it to Raspberry

Pi board which has also been completed. For this purpose a make file compatible to Linux environment for the code was generated and an executable was created.

### 3.2.2 AES Implementation

In this step, using the same resources as that in serial communication task, AES was first implemented in windows, then in Linux and ultimately ran on Raspberry Pi board.

### 3.3 Detailed design

In our design, the first step includes transmission of data from one PC to other via RS232 standard using RTS handshaking. In the next step AES c/c++ code would be used to encrypt/decrypt data and that very encrypted data would be shared between PCs. In the next sequence of events, the same steps would be imported to hardware cubie board with Linux OS in it. Once the module would be fully functional and capable of encrypting and sending the data, it will be tested with the wireless radio set. A general flowchart at the end of our project showing sequence of events taking place in module is given below:

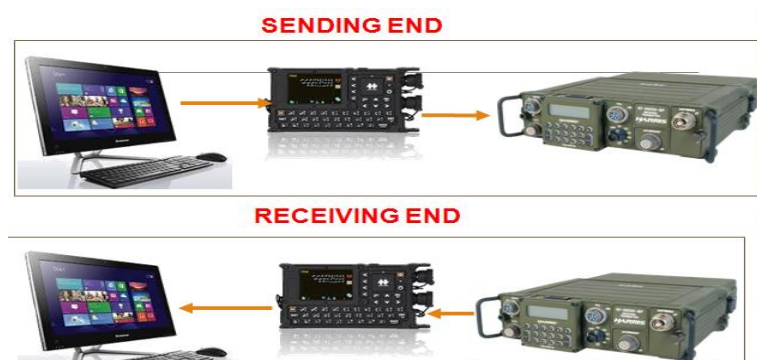


Figure 3-6: Detailed Design

On the Sending end the user enters data in the command window with send script running on it, the data is encrypted automatically and sent to radio connected to the module. Radio sends this cipher text of original data to another radio which hands it over to second module on receiving end. This module then decrypts the data and displays the message to the user at receiving end sent by the user at sending end.

### 3.4 Tasks and Challenges

The project had some main tasks and challenges which included development of code for communication between two computers then searching for perfect encryption technique

and choosing the best and affordable hardware to Implement Encryption. After selection of all the main things, major task was to implement it with as little resources as possible and make it as user friendly as possible for example keeping the delay as low as possible. Next task was to implement the encryption technique on to the hardware and to ensure communication between add on security module and radio Set.

## 4. Project Analysis and Evaluation

This chapter focuses on final development of module including the methodology used and results achieved. It also briefs about the method used for testing and narrates how good the performance of the module designed was.

The aim of our project was to design a module that could be used by soldiers in battlefield to encrypt the message before transmitting it wirelessly using hf set and to also decrypt the encrypted message received on module from other users. The project was tested for RF5800h MP, a military wireless set.

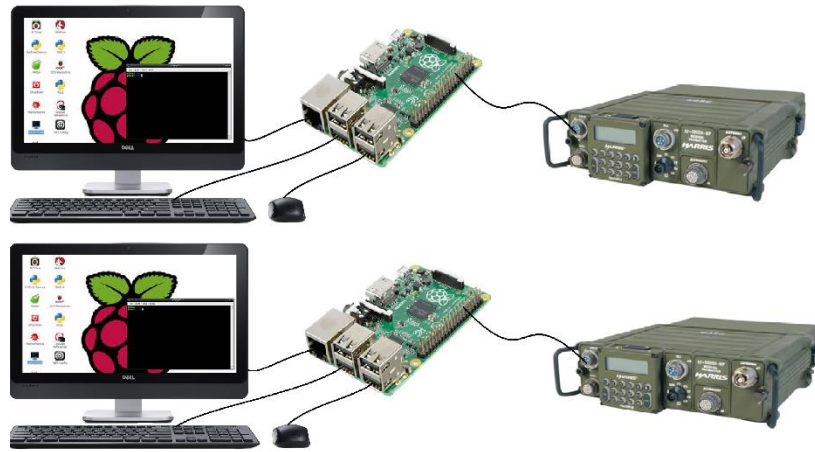


Figure 4-1: Finalized Design

### 4.1 AES implementation

One of the main tasks of the project was implementation of AES algorithm in raspberry pi board with effective use of key and as little delay as possible. The code for AES was developed using c language and was compiled using standard gcc compiler of Raspberry pi board. In the first sequence of events on encrypting side, the user was asked to provide a password which was then automatically stored in .txt file format in the working directory of program. The hash of the file with contents provided by user as password was then created using SHA-1 algorithm, also developed in c language using g++ compiler. After successful generation of hash file, the user was asked to enter the text to be encrypted using the password provided by the user. The text was encrypted with hash of user entered password using AES-256. The delay for the complete encryption was set to be 500ms which is quite economical and acceptable for the user.

On the decrypting side, same password used for encryption was used to generate another hash file using SHA-1 algorithm. The both hashes i.e. that of received file and the one generated using password in the same module, were then compared and on successful comparison the cipher text was decrypted and displayed on screen to the user.

## **4.2 Serial Communication**

Next task ahead was the development of the code for serial communication on Raspberry pi board. The code was developed in c++ language using wiringPi library and compiled using g++ compiler of raspberry Pi. The UART was developed using GPIO pins of raspberry pi. Baud rate was set to be 19200bps to match the baud rate of radio set used for testing. To test the successful running of codes a null modem was established between two Raspberry Pi and a non-erroneous two way communication was achieved successfully with a delay of 20ms.

## **4.3 Combining AES and Serial Communication**

In the final step, code for AES and Serial communication were made to work together using bash programming. Two separate scripts for sending and receiving were developed. The script for send asked the user for data whenever the RTS pin was set, encrypted it and transmitted it serially to the radio set. Whereas the receiving script received the cipher text, decrypted it and displayed it to the user. In this way a secure tactical chat application was developed. Baud rate was again kept to be 19200Hz for serial communication.

## **4.4 Final testing**

The final design included two raspberry pi boards with fully developed AES and Serial communication codes connected to Radios using Null Modem and TTL to RS232 converters between Pi boards and radios. The setup was so that gpio transmitting and receiving pins of one of the module were connected to TTL to RS232 logic converter circuit which was then connected to radio via Null modem.

Same connections were made for second module with another radio set. The two radio sets were then configured over a single RF channel and tested for communication without modules. After the successful wireless communication between radio sets, the modules were used to send and receive data over the radio sets. The timeout for the whole procedure was set to be 750ms and a data file of up to 80k bytes was required to be sent. The goal

was achieved.

## 4.5 Data Flow Model

### **Sending End:**

The flow of the data in the sending end of module starts from getting input from the user which can either be data or text file. The input is then encrypted using AES-256 and store in a new unreadable file. The AES file is then handed over to radio by the security module which wirelessly transmits it over the frequency for which it is configured. Any of the both modules as well as radios can act as sender or receiver depending upon the current state of respective RTS pin of radio or the current script i.e. that of sending or receiving used by the user.

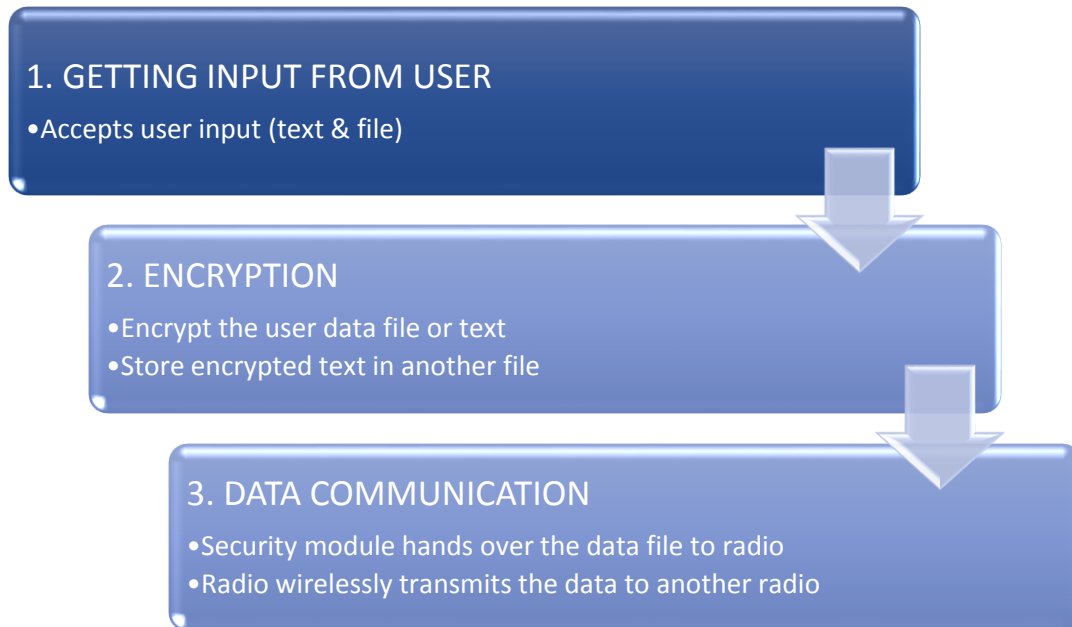


Figure 4-2: Data Flow Model (sending end)

### **Receiving end:**

The flow of the data in the receiving end of module starts from getting input from the reception of encrypted file by the other radio. This encrypted file is handed over to the other module by the radio set. The module after receiving encrypted file from radio, decrypts it and stores the readable decrypted file in .txt format. After successfully saving the file in directory it is also shown on screen to the user in the form of text in tactical chat window.

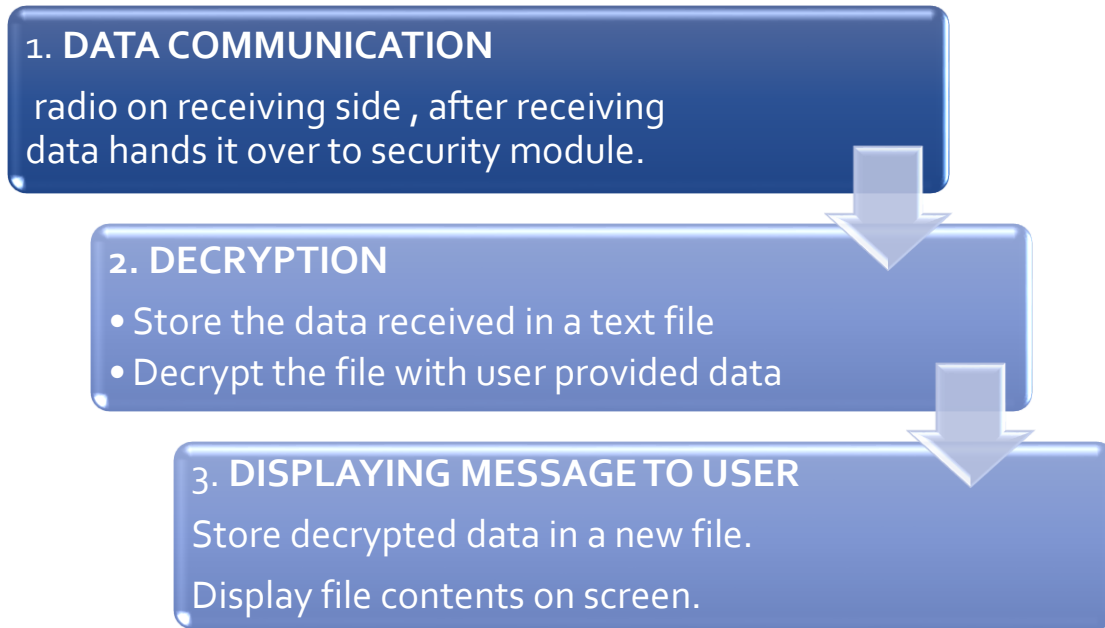


Figure 4-3: Data Flow Model (receiving end)

All the Objectives proposed in the beginning were thus met and verified.



## **5 Recommendations and Conclusion**

The chapter briefs the reader about the current status of the project, what are its limitations and how this can be improved in future.

### **5.1 Overview**

The main idea of the project was to design and create such a security module which can be connected to radio sets externally and shall still be completely functional and useful in terms of providing the security and preserving confidentiality of data. The project was research based and aimed at checking the feasibility of the idea of addition of external security module for overcoming hazard of lack of trust on foreign origin based security algorithms.

### **5.2 Limitations**

Since before starting it, the project was just an idea so we were given a task to implement idea only for simple text data since data requires no real time processing like compression techniques. So, the security module has only been developed and tested for simple text data.

Also, the data port of radio used for testing purpose can only send and receive files up to 80 kilo bytes so the data larger than this limit could not be tested.

### **5.3 Recommendations**

The current module focuses on securing data and text files only. Also, it is only developed to be used with data ports of the radios. In future, students can further extend the project by making it compatible with all sorts of data like voice, video. By adding a codec and some compression techniques in the code the student shall be able to implement the project for voice as well. The security module can also be developed IP data and for radio sets other than HF sets.

### **5.4 Conclusion**

During recent times, AES and its implementation has become something substantially used in all kinds of projects involving communication and information sharing. But no work on AES involving radio sets especially those used by soldiers has been done yet in any of the students' projects. The module we have created is first of its kind and it will open up doors of new field in projects. Overall, the

objectives of the project were met. The design process of module was examined. The module was also tested with a radio set namely RF-5800h MP to check for its application based use.

## BIBLIOGRAPHY

## 6. Bibliography

### 6.1 General References

[1]Margaret Rouse (may 10, 2016). *Advanced Encryption Standard (AES)*. Retrieved from <http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard/>

[2]Anitha, P. & Palanisamy, V. (2011) *Data protection algorithm using AES*. International journal of current research, 3(6), pp. 291-294,

[3]Beer, K. & Holland, R. Amazon web services. (2013). *Securing Data at Rest with Encryption*. Washington, DC: U.S. Amazon Printing Office

[4]Hollabaugh, C. (2004) *Embedded Linux: Hardware, Software and Interfacing*. Boston, Massachusetts: Addison Wesley.

[5]Breese, F. (2010) *Serial Communication Over RTP/CDP*. Norderstedt, Germany: books on demand.

[6]Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Fotti, M. & Edward Roback. (2000) *Report on the Development of the Advanced Encryption Standard (AES)*. Retrieved from Computer Security Division, National Institute of Standards and Technology, U.S. Department of Commerce. <http://csrc.nist.gov/archive/aes/round2/r2report.pdf>

### 6.2 Online Help

[1]AES 128 bit Encryption <http://aesencryption.net/> retrieved on Oct 19, 2015.

[2] Harris Equipment “Radio set RF-5800h-mp” <http://rf.harris.com/capabilities/tactical-radios-networking/rf-5800h-mp.asp>, retrieved on Oct 11, 2015.

### 6.3 In text Citations

[1][http://www.silabs.com/Support%20Documents/Software/Serial\\_Communications.pdf](http://www.silabs.com/Support%20Documents/Software/Serial_Communications.pdf) retrieved in August, 2016

[2] *Computer Security Resource Center*. (2016, June 9). Retrieved from National Institute of Standards and Technolo

[3]Kretzschmar, U. (2009). *AES128 – A C Implementation for Encryption and decrytion*. Texas: Texas Instruments

[4]Mahanta, K., & Maringanti, H. B. (2015). An Enhanced Advanced Encryption Standard Algorithm. *International Journal of Advanced Trends in Computer Science and Engineering* , 4 (4), 28-33

[5]Eastlake, D., Motorola, & Jones, P. E. (2001, September). *US secure hash algorithm 1 (SHA1)*. Retrieved from <https://tools.ietf.org/html/rfc3174>. Retrieved on June 16, 2016.

[6]Secure hash standard. Retrieved from <http://cis-linux1.temple.edu/~giorgio/cis307/readings/sha1.html> on Jan 10, 2016

[7]Tech stuff - RS232 cables and wiring. (2016, April 06). Retrieved from [http://www.zytrax.com/tech/layer\\_1/cables/tech\\_rs232.htm](http://www.zytrax.com/tech/layer_1/cables/tech_rs232.htm)

[8]Richardson, M., & Wallace, S. (2012). *Getting Started with Raspberry Pi*. O'Reilly Media, Inc.

[9]Jimb0. (2010, November 23). *RS-232 vs. TTL Serial Communication*. Retrieved from Sparkfun: <https://www.sparkfun.com/tutorials/215>. Retrieved on June 16, 2016.

[10]Code for AES retrieved from [www.github.com](http://www.github.com) in December, 2015

## APPENDIX

## APPENDIX-A

### CODES

## CODE FOR ENCRYPTION/DECRYPTION

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <time.h>
#include "aescrypt.h"
#include "password.h"
#include "keyfile.h"
#include "util.h"
#include "aesrandom.h"
int encrypt_stream(FILE *infp, FILE *outfp, unsigned char* passwd, int passlen)
{
    aes_context aes_ctx;
    sha256_context sha_ctx;
    aescrypt_hdr aeshdr;
    sha256_t digest;
    unsigned char IV[16];
    unsigned char iv_key[48];
    unsigned i, j;
    size_t bytes_read;
    unsigned char buffer[32];
    unsigned char ipad[64], opad[64];
    time_t current_time;
    pid_t process_id;
    void *aesrand;
    unsigned char tag_buffer[256];
    memset(iv_key, 0, 48);
    for (i=0; i<48; i+=16)
    {
        memset(buffer, 0, 32);
        sha256_starts(&sha_ctx);
        for(j=0; j<256; j++)
        {
            if ((bytes_read = aesrandom_read(aesrand, buffer, 32)) != 32)
            {
                fprintf(stderr, "Error: Couldn't read from random : %u\n",
                    (unsigned) bytes_read);
            }
        }
    }
}
```



```

aesrandom_close(aesrand);
return -1;
}
sha256_update(&sha_ctx, buffer, 32);
}
sha256_finish(&sha_ctx, digest);
memcpy(iv_key+i, digest, 16);
}
buffer[0] = 'A';
buffer[1] = 'E';
buffer[2] = 'S';
buffer[3] = (unsigned char) 0x02; /* Version 2 */
buffer[4] = '\0'; /* Reserved for version 0 */
if (fwrite(buffer, 1, 5, outfp) != 5)
{
fprintf(stderr, "Error: Could not write out header data\n");
aesrandom_close(aesrand);
return -1;
}
if (j < 256)
{
buffer[0] = '\0';
buffer[1] = (unsigned char) (j & 0xff);
if (fwrite(buffer, 1, 2, outfp) != 2)
{
fprintf(stderr, "Error: Could not write tag to AES file (1)\n");
aesrandom_close(aesrand);
return -1;
}
strncpy((char *)tag_buffer, "CREATED_BY", 255);
tag_buffer[255] = '\0';
if (fwrite(tag_buffer, 1, 11, outfp) != 11)
{
fprintf(stderr, "Error: Could not write tag to AES file (2)\n");
aesrandom_close(aesrand);
return -1;
}
sprintf((char *)tag_buffer, "%s %s", PACKAGE_NAME, PACKAGE_VERSION);
j = strlen((char *)tag_buffer);
if (fwrite(tag_buffer, 1, j, outfp) != (size_t)j)
{
fprintf(stderr, "Error: Could not write tag to AES file (3)\n");
aesrandom_close(aesrand);
return -1;
}
}
}
}

```

```

/* Write out the "container" extension */
buffer[0] = '\0';
buffer[1] = (unsigned char) 128;
if (fwrite(buffer, 1, 2, outfp) != 2)
{
    fprintf(stderr, "Error: Could not write tag to AES file (4)\n");
    aesrandom_close(aesrand);
    return -1;
}
memset(tag_buffer, 0, 128);
if (fwrite(tag_buffer, 1, 128, outfp) != 128)
{
    fprintf(stderr, "Error: Could not write tag to AES file (5)\n");
    aesrandom_close(aesrand);
    return -1;
}
/* Write out 0x0000 to indicate that no more extensions exist */
buffer[0] = '\0';
buffer[1] = '\0';
if (fwrite(buffer, 1, 2, outfp) != 2)
{
    fprintf(stderr, "Error: Could not write tag to AES file (6)\n");
    aesrandom_close(aesrand);
    return -1;
}
current_time = time(NULL);
for(i = 0; i < 8; i++)
{
    buffer[i] = (unsigned char)
(current_time >> (i * 8));
}
process_id = getpid();
for(i = 0; i < 8; i++)
{
    buffer[i+8] = (unsigned char)
(process_id >> (i * 8));
}
sha256_starts( &sha_ctx);
sha256_update( &sha_ctx, buffer, 16);
for (i=0; i<256; i++)
{
    if (aesrandom_read(aesrand, buffer, 32) != 32)
    {
        fprintf(stderr, "Error: Couldn't read from /dev/random\n");
        aesrandom_close(aesrand);
        return -1;
    }
}

```

```

}
sha256_update( &sha_ctx,
buffer,
32);
}
sha256_finish( &sha_ctx, digest);
memcpy(IV, digest, 16);
aesrandom_close(aesrand);

if (fwrite(IV, 1, 16, outfp) != 16)
{
fprintf(stderr, "Error: Could not write out initialization vector\n");
return -1;
}
/* Hash the IV and password 8192 times */
memset(digest, 0, 32);
memcpy(digest, IV, 16);
for(i=0; i<8192; i++)
{
sha256_starts( &sha_ctx);
sha256_update( &sha_ctx, digest, 32);
sha256_update( &sha_ctx,
passwd,
(unsigned long)passlen);
sha256_finish( &sha_ctx,
digest);
}
aes_set_key(&aes_ctx, digest, 256);
memset(ipad, 0x36, 64);
memset(opad, 0x5C, 64);
for(i=0; i<32; i++)
{
ipad[i] ^= digest[i];
opad[i] ^= digest[i];
}
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, ipad, 64);
for(i=0; i<48; i+=16)
{
memcpy(buffer, iv_key+i, 16);
for(j=0; j<16; j++)
{
buffer[j] ^= IV[j];
}
aes_encrypt(&aes_ctx, buffer, buffer);
sha256_update(&sha_ctx, buffer, 16);

```

```

if (fwrite(buffer, 1, 16, outfp) != 16)
{
fprintf(stderr, "Error: Could not write iv_key data\n");
return -1;
}
memcpy(IV, buffer, 16);
}
sha256_finish(&sha_ctx, digest);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, opad, 64);
sha256_update(&sha_ctx, digest, 32);
sha256_finish(&sha_ctx, digest);
if (fwrite(digest, 1, 32, outfp) != 32)
{
fprintf(stderr, "Error: Could not write iv_key HMAC\n");
return -1;
}
memcpy(IV, iv_key, 16);
aes_set_key(&aes_ctx, iv_key+16, 256);
memset(ipad, 0x36, 64);
memset(opad, 0x5C, 64);
for(i=0; i<32; i++)
{
ipad[i] ^= iv_key[i+16];
opad[i] ^= iv_key[i+16];
}
memset_secure(iv_key, 0, 48);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, ipad, 64);
aeshdr.last_block_size = 0;
while ((bytes_read = fread(buffer, 1, 16, infp)) > 0)
{
for(i=0; i<16; i++)
{
buffer[i] ^= IV[i];
}
aes_encrypt(&aes_ctx, buffer, buffer);
sha256_update(&sha_ctx, buffer, 16);
if (fwrite(buffer, 1, 16, outfp) != 16)
{
fprintf(stderr, "Error: Could not write to output file\n");
return -1;
}
memcpy(IV, buffer, 16);
aeshdr.last_block_size = bytes_read;
}

```

```

if (ferror(infp))
{
fprintf(stderr, "Error: Couldn't read input file\n");
return -1;
}
buffer[0] = (char) (aeshdr.last_block_size & 0x0F);
if (fwrite(buffer, 1, 1, outfp) != 1)
{
fprintf(stderr, "Error: Could not write the file size modulo\n");
return -1;
}
sha256_finish(&sha_ctx, digest);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, opad, 64);
sha256_update(&sha_ctx, digest, 32);
sha256_finish(&sha_ctx, digest);
if (fwrite(digest, 1, 32, outfp) != 32)
{
fprintf(stderr, "Error: Could not write the file HMAC\n");
return -1;
}
if (fflush(outfp))
{
fprintf(stderr, "Error: Could not flush output file buffer\n");
return -1;
}
return 0;
}
int decrypt_stream(FILE *infp, FILE *outfp, unsigned char* passwd, int passlen)
{
aes_context aes_ctx;
sha256_context sha_ctx;
aescrypt_hdr aeshdr;
sha256_t digest;
unsigned char IV[16];
unsigned char iv_key[48];
unsigned i, j, n;
size_t bytes_read;
unsigned char buffer[64], buffer2[32];
unsigned char *head, *tail;
unsigned char ipad[64], opad[64];
int reached_eof = 0;
/* Read the file header */
if ((bytes_read = fread(&aeshdr, 1, sizeof(aeshdr), infp)) !=
sizeof(aescrypt_hdr))
{

```

```

if (feof(infp))
{
fprintf(stderr, "Error: Input file is too short.\n");
}
else
{
perror("Error reading the file header:");
}
return -1;
}
if (!(aeshdr.aes[0] == 'A' && aeshdr.aes[1] == 'E' &&
aeshdr.aes[2] == 'S'))
{
fprintf(stderr, "Error: Bad file header (not aescrypt file or is corrupted? [%x, %x,
%x])\n", aeshdr.aes[0], aeshdr.aes[1], aeshdr.aes[2]);
return -1;
}
aeshdr.last_block_size = (aeshdr.last_block_size & 0x0F);
}
else if (aeshdr.version > 0x02)
{
fprintf(stderr, "Error: Unsupported AES file version: %d\n",
aeshdr.version);
return -1;
}
if (aeshdr.version >= 0x02)
{
do
{
if ((bytes_read = fread(buffer, 1, 2, infp)) != 2)
{
if (feof(infp))
{
fprintf(stderr, "Error: Input file is too short.\n");
}
else
{
perror("Error reading the file extensions:");
}
return -1;
}
i = j = (((int)buffer[0]) << 8) | (int)buffer[1];
while (i--)
{
if ((bytes_read = fread(buffer, 1, 1, infp)) != 1)
{

```



```

}
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, ipad, 64);
if (aeshdr.version >= 0x01)
{
for(i=0; i<48; i+=16)
{
if ((bytes_read = fread(buffer, 1, 16, infp)) != 16)
{
if (feof(infp))
{
fprintf(stderr, "Error: Input file is too short.\n");
}
else
{
perror("Error reading input file IV and key:");
}
return -1;
}
memcpy(buffer2, buffer, 16);
sha256_update(&sha_ctx, buffer, 16);
aes_decrypt(&aes_ctx, buffer, buffer);
for(j=0; j<16; j++)
{
iv_key[i+j] = (buffer[j] ^ IV[j]);
}
memcpy(IV, buffer2, 16);
}
sha256_finish(&sha_ctx, digest);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, opad, 64);
sha256_update(&sha_ctx, digest, 32);
sha256_finish(&sha_ctx, digest);
if ((bytes_read = fread(buffer, 1, 32, infp)) != 32)
{
if (feof(infp))
{
fprintf(stderr, "Error: Input file is too short.\n");
}
else
{
perror("Error reading input file digest:");
}
return -1;
}
if (memcmp(digest, buffer, 32))

```



```

{
fprintf(stderr, "Error: Message has been altered or password is incorrect\n");
return -1;
}
memcpy(IV, iv_key, 16);
aes_set_key(&aes_ctx, iv_key+16, 256);
memset(ipad, 0x36, 64);
memset(opad, 0x5C, 64);
for(i=0; i<32; i++)
{
ipad[i] ^= iv_key[i+16];
opad[i] ^= iv_key[i+16];
}
memset_secure(iv_key, 0, 48);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, ipad, 64);
}
if ((bytes_read = fread(buffer, 1, 48, infp)) < 48)
{
if (!feof(infp))
{
perror("Error reading input file ring:");
return -1;
}
else
{
if ((aeshdr.version == 0x00 && bytes_read != 32) ||
(aeshdr.version >= 0x01 && bytes_read != 33))
{
fprintf(stderr, "Error: Input file is corrupt (1:%u).\n",
(unsigned) bytes_read);
return -1;
}
else
{
if (aeshdr.version >= 0x01)
{

aeshdr.last_block_size = (buffer[0] & 0x0F);
}
if (aeshdr.last_block_size != 0)
{
fprintf(stderr, "Error: Input file is corrupt (2).\n");
return -1;
}
}
}
}

```

```

reached_eof = 1;
}
}
head = buffer + 48;
tail = buffer;
while(!reached_eof)
{
if (head == (buffer + 64))
{
head = buffer;
}
if ((bytes_read = fread(head, 1, 16, infp)) < 16)
{
if (!feof(infp))
{
perror("Error reading input file:");
return -1;
}
else
{
if ((aeshdr.version == 0x00 && bytes_read > 0) || (aeshdr.version >= 0x01 &&
bytes_read != 1))
{
fprintf(stderr, "Error: Input file is corrupt (3:%u).\n",
(unsigned) bytes_read);
return -1;
}
if (aeshdr.version >= 0x01)
{
if ((tail + 16) < (buffer + 64))
{
aeshdr.last_block_size = (tail[16] & 0x0F);
}
else
{
aeshdr.last_block_size = (buffer[0] & 0x0F);
}
}
}
reached_eof = 1;
}
}
if ((bytes_read > 0) || (aeshdr.version == 0x00))
{
if (bytes_read > 0)
{
head += 16;

```

```

}
memcpy(buffer2, tail, 16);
sha256_update(&sha_ctx, tail, 16);
aes_decrypt(&aes_ctx, tail, tail);
for(i=0; i<16; i++)
{
tail[i] ^= IV[i];
}
memcpy(IV, buffer2, 16);
n = ((!reached_eof) ||
(aeshdr.last_block_size == 0)) ? 16 : aeshdr.last_block_size;
/* Write the decrypted block */
if ((i = fwrite(tail, 1, n, outfp)) != n)
{
perror("Error writing decrypted block:");
return -1;
}
/* Move the tail of the ring buffer forward */
tail += 16;
if (tail == (buffer+64))
{
tail = buffer;
}
}
sha256_finish(&sha_ctx, digest);
sha256_starts(&sha_ctx);
sha256_update(&sha_ctx, opad, 64);
sha256_update(&sha_ctx, digest, 32);
sha256_finish(&sha_ctx, digest);
if (aeshdr.version == 0x00)
{
memcpy(buffer2, tail, 16);
tail += 16;
if (tail == (buffer + 64))
{
tail = buffer;
}
memcpy(buffer2+16, tail, 16);
}
else
{
memcpy(buffer2, tail+1, 15);
tail += 16;
if (tail == (buffer + 64))
{

```

```

tail = buffer;
}
memcpy(buffer2+15, tail, 16);
tail += 16;
if (tail == (buffer + 64))
{
tail = buffer;
}
memcpy(buffer2+31, tail, 1);
}
if (memcmp(digest, buffer2, 32))
{
if (aeshdr.version == 0x00)
{
fprintf(stderr, "Error: Message has been altered or password is incorrect\n");
}
else
{
fprintf(stderr, "Error: Message has been altered and should not be trusted\n");
}
return -1;
}
if (fflush(outfp))
{
fprintf(stderr, "Error: Could not flush output file buffer\n");
return -1;
}
return 0;
}
int main(int argc, char *argv[])
{
int rc=0;
int passlen=0;
FILE *infp = NULL;
FILE *outfp = NULL;
encryptmode_t mode=UNINIT;
char *infile = NULL;
unsigned char pass[MAX_PASSWD_BUF];
int file_count = 0;
char outfile[1024];
int password_acquired = 0;
/* Initialize the output filename */
outfile[0] = '\0';
while ((rc = getopt(argc, argv, "vhdek:p:o:")) != -1)
{
switch (rc)

```

```

{
case 'h':
usage(argv[0]);
return 0;
case 'v':
version(argv[0]);
return 0;
case 'd':
if (mode != UNINIT)
{
fprintf(stderr, "Error: only specify one of -d or -e\n");
cleanup(outfile);
return -1;
}
mode = DEC;
break;
case 'e':
if (mode != UNINIT)
{
fprintf(stderr, "Error: only specify one of -d or -e\n");
cleanup(outfile);
return -1;
}
mode = ENC;
break;
case 'k':
if (password_acquired)
{
fprintf(stderr, "Error: password supplied twice\n");
cleanup(outfile);
return -1;
}
if (optarg != 0)
{
if (!strcmp("-",optarg))
{
fprintf(stderr,
"Error: keyfile cannot be read from stdin\n");
cleanup(outfile);
return -1;
}
passlen = ReadKeyFile(optarg, pass);
if (passlen < 0)
{
cleanup(outfile);
return -1;
}
}

```

```

}
password_acquired = 1;
}
break;
case 'p':
if (password_acquired)
{
fprintf(stderr, "Error: password supplied twice\n");
cleanup(outfile);
return -1;
}
if (optarg != 0)
{
passlen = passwd_to_utf16( (unsigned char*) optarg,
strlen((char *)optarg),
MAX_PASSWD_LEN,
pass);
if (passlen < 0)
{
cleanup(outfile);
return -1;
}
password_acquired = 1;
}
break;
case 'o':
/* outfile argument */
if (!strcmp("-", optarg, 2))
{
/* if '-' is outfile name then out to stdout */
outfp = stdout;
}
else if ((outfp = fopen(optarg, "w")) == NULL)
{
fprintf(stderr, "Error opening output file %s:", optarg);
perror("");
cleanup(outfile);
return -1;
}
strncpy(outfile, optarg, 1024);
outfile[1023] = '\0';
break;
default:
fprintf(stderr, "Error: Unknown option '%c'\n", rc);
}
}

```

```

if (optind >= argc)
{
fprintf(stderr, "Error: No file argument specified\n");
usage(argv[0]);
cleanup(outfile);
return -1;
}
if (mode == UNINIT)
{
fprintf(stderr, "Error: -e or -d not specified\n");
usage(argv[0]);
cleanup(outfile);
return -1;
}
/* Prompt for password if not provided on the command line */
if (passlen == 0)
{
passlen = read_password(pass, mode);
switch (passlen)
{
case 0: /* no password in input */
fprintf(stderr, "Error: No password supplied.\n");
cleanup(outfile);
return -1;
case AESCRYPT_READPWD_FOPEN:
case AESCRYPT_READPWD_FILENO:
case AESCRYPT_READPWD_TCGETATTR:
case AESCRYPT_READPWD_TCSETATTR:
case AESCRYPT_READPWD_FGETC:
case AESCRYPT_READPWD_TOOLONG:
case AESCRYPT_READPWD_ICONV:
fprintf(stderr, "Error in read_password: %s.\n",
read_password_error(passlen));
cleanup(outfile);
return -1;
case AESCRYPT_READPWD_NOMATCH:
fprintf(stderr, "Error: Passwords don't match.\n");
cleanup(outfile);
return -1;
}
}
file_count = argc - optind;
if ((file_count > 1) && (outfp != NULL))
{
if (outfp != stdout)
{

```

```

fclose(outfp);
}
fprintf(stderr, "Error: A single output file may not be specified with multiple input
files.\n");
usage(argv[0]);
cleanup(outfile);
/* For security reasons, erase the password */
memset_secure(pass, 0, MAX_PASSWD_BUF);
return -1;
}
if (mode == ENC)
{
if (outfp == NULL)
{
snprintf(outfile, 1024, "%s.aes", infile);
if ((outfp = fopen(outfile, "w")) == NULL)
{
if ((infp != stdin) && (infp != NULL))
{
fclose(infp);
}
fprintf(stderr, "Error opening output file %s : ", outfile);
perror("");
cleanup(outfile);
/* For security reasons, erase the password */
memset_secure(pass, 0, MAX_PASSWD_BUF);
return -1;
}
}
rc = encrypt_stream(infp, outfp, pass, passlen);
}
else if (mode == DEC)
{
if (outfp == NULL)
{
strncpy(outfile, infile, strlen(infile)-4);
outfile[strlen(infile)-4] = '\0';
if ((outfp = fopen(outfile, "w")) == NULL)
{
if ((infp != stdin) && (infp != NULL))
{
fclose(infp);
}
fprintf(stderr, "Error opening output file %s : ", outfile);
perror("");
cleanup(outfile);
}
}
}
}

```



```
}  
}}[10]
```

## CODES FOR SERIAL COMMUNICATION

### 1. Code to Get Input From User:

```
#include <iostream>  
  
#include <fstream>  
  
using namespace std;  
  
int main()  
{  
    char data[100];  
    ofstream outfile;  
    outfile.open("text.txt");  
    cout << "Enter text you want to encrypt: ";  
    cin.getline(data,100);  
    outfile << data << endl;  
    outfile.close();}
```

### Code to serially receive the data:

```
#include <iostream>  
  
#include <fstream>  
  
#include <wiringSerial.h>  
  
#include <stdlib.h>  
  
#include <unistd.h>  
  
using namespace std;  
  
int main()  
{  
    ofstream outfile;  
    outfile.open("txt1.txt.aes",ios::binary);  
    int fd = serialOpen("/dev/ttyAMA0",19200);  
    if (fd==-1)  
    {  
        cout<< " Serial Error" << endl;  
        return 0;    }  
  
    char ch;
```

```

while (1)
{
    if (serialDataAvail(fd)>0)
        {
            while (serialDataAvail(fd)>0)
                {
                    read(fd,&ch,1);
                    outfile.write(&ch,1);
                    usleep(500);  }
            outfile.close();
            serialClose(fd);
            return 0;    }}}

```

### **Code to Send Data Serially:**

```

#include <iostream>
#include <fstream>
#include <wiringSerial.h>
#include <unistd.h>
using namespace std;
int main()
{
    int fd;
    fd=serialOpen("/dev/ttyAMA0",19200);
    if (fd==-1)
        {
            cout << "Serial not open";
            return 0;    }
    ifstream infile;
    infile.open("text.txt.aes",ios::binary);
    infile.seekg(0,infile.end);
    int length = infile.tellg();
    infile.seekg(0,infile.beg);
    char ch;
    while (length>0)
        {
            infile.read(&ch,1);
            write(fd,&ch,1);

```

```
        length--;}  
infile.close();  
serialClose(fd);}
```