Wireless USB Access – An Intermediate between USB and RF Technology

By

NC Ammar Amir Khan

PC Danial Sadiq

PC Muhammad Suleman Khan

Submitted to the Faculty of Department of Electrical Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad
In partial fulfillment for the requirements of a B.E Degree in
Telecom Engineering July, 2015

CERTIFICATE OF CORRECTNESS AND APPROVAL

Certified that work contained in the thesis "Wireless USB Access – An Intermediate between USB & RF Technology", was carried out by NC Ammar Amir, PC Danial Sadiq, and PC Muhammad Suleman under the supervision of Lt. Col Imran Rashid for partial fulfilment of Degree of Bachelor of Telecommunication Engineering, is correct and approved.

Approved by

_____

(Lt. Col. Dr. Imran Rashid)

Project Directing Staff (DS)

Military College of Signals (MCS), National University of Sciences and Technology (NUST)

Dated: _____ June, 2015

ABSTRACT

WIRELESS USB ACCESS – AN INTERMEDIATE BETWEEN USB AND RF TECHNOLOGY

USB flash devices – have seen a large influx in consumer use in recent years – especially with reference to sensitive data. With increasing consumer demands for prompt data accessibility, technologists have introduced the concept of wireless storage. During the past few years, advancements have mainly been focused on internet-based wireless storage solutions (Cloud Storage); however with the vulnerabilities of the internet as a whole from the diversely persistent hacker community, storing sensitive personal/business data online is no longer a viable solution. Bearing these problems in mind, we decided to contrive a solution that will enable flash storage data to securely be made accessible on common Bluetooth enabled smart devices. In a nutshell, our project is a storage medium interfaced to a Bluetooth module via controller along with AES encryption/decryption. We propose our solution as an alternative to other wireless data storage solutions.

DEDICATION

ALMIGHTY ALLAH

FACULTY FOR THEIR SUPPORT,

AND OUR PARENTS FOR THEIR PRAYERS.

ACKNOWLEDGEMENTS

We express our total submission and eternal gratitude to ALMIGHTY ALLAH, The most Gracious, The Most Merciful, without mercy and grace, this endeavor could not been possible even in its slightest. We deem it an honor and privilege to pay our heartfelt thanks and gratitude to our project supervisor Lt. Col. Dr. Imran Rashid for his guidance and continuous encouragement and inspiration, especially during hard and desperate phases of our work. He helped us every step of the way in completing this tedious work.

Also, we would like to extend our whole hearted thanks to the faculty members of the Electrical Engineering Department, for providing necessary help and guidance during this project.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

USB	Universal Serial Bus
MSD	Mass Storage Device
AES	Advanced Encryption Standard
RF	Radio Frequency
ISM Band	Industrial, Scientific & Medical Band
S-box	Substitution Box
SD Card	Secure Digital Card

Chapter 1: Introduction

Wireless USB Access is a project aimed at providing secure alternatives to current available data access techniques. We intend to contrive a secure interface between the standard USB storage protocols and Bluetooth technologies.

Since its inception, Universal Serial Bus (USB) has become the standard peripheral interface in personal computers. Research statistics[1] indicate USB flash drives to be a popular storage device amongst consumers. But with the evolution of technology our sensitive data is more prone to hacking. Hence, we try to solve the problem by designing and developing an interface between Bluetooth and USB storage targeting a prototype device which shall act as a storage device using RF technology (Bluetooth).

1.1 Project Description & Salient Features:

Our project comprises of 3 parts; a controlling module (Teensy 3.1), a Bluetooth module (Linvor HC-05), and a flash storage medium (microSD card). The data is being accessed on our smartphone devices via an android application. The controlling module is the most important component of our project; it integrates all the hardware modules together by serving as the main controlling and data processing unit. The module encrypts data before sending the information to the Bluetooth IC. Furthermore, the controlling module is also responsible of decrypting the data upon receiving it from the smart phone before storing it in the SD card. Secondary to the Teensy, the HC-05 is being used to provide the radio-transmission facility. Linvor Bluetooth products – in particular the HC series – have been used in a diverse array of Android and Arduino based projects, furthering its relative applicability for the specific case of our project.

The third part of our project is the SD card. This is used to store the information received by the controller from the smartphone. The SD card is interfaced with the controller such that the data movement is bi-directional. The received data can be stored and accessed when required.

[1] Statistical survey carried out by SanDisk® [1].

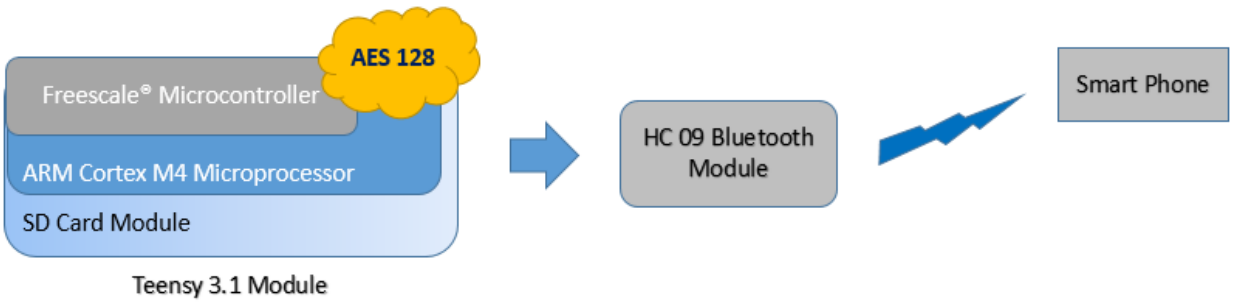


Figure 1 : Project Model

1.2 Prospective Applications

Our solution is intended for consumers looking for secure wireless storage alternatives which serve their requirements of accessing their data on the move. With the advent of new industries – including m-commerce and other industrial systems, consumers are increasingly looking for solutions where they can access their data wirelessly whilst being assured that the integrity of their sensitive information remains intact. Our project is not targeted towards private or individual consumers. Instead it is targeted towards large conglomerates and other security organizations that demand additional levels of information security. This is due to the adaptive and customizable nature of our project. By virtue of our onboard programmable controller, it is easy to install any customized encryption scheme in our solution. The random nature of the frequency hopping spread spectrum adopted by the Bluetooth protocol alongside the limited transmission range further strengthens the security levels provided through this project.

1.3 Objectives

Our project objective is to solve a real-life problem by designing and developing a secure wireless storage device usable by non-technical market consumers. In the process, we will achieve the following academic objectives:

- Embedded Systems Design and Implementation (Programming and Interfacing on ARM Cortex M4).
- Wireless Communication Protocols and Troubleshooting (Bluetooth on the HC-05 Bluetooth Module).
- Practical Industrial Circuit Design, Development and Integration Techniques.
- Modern Information Security Techniques with Advanced Encryption Standard (AES).

We aim to employ our knowledge of telecommunications' engineering to best exploit our resources to produce a working wireless storage prototype.

1.4 Requirements, Scope, Specification, and Deliverables

Bearing in mind the strict contingencies of our project, our team conducted a thorough requirements analysis. Our hardware selection is based on the outcomes of that analysis. As previously mentioned, our project is comprised of three main aspects; namely Bluetooth integration, information security, and data storage.

After much deliberating research, we have chosen to employ the Linvor HC-05 Bluetooth Module to provide the Radio-Transmission Facility. Of the whole microwave spectrum, the ISM Band seems the most appealing option, as it falls under The Deregulated Frequency Band. Bluetooth (a predominant protocol of the ISM band) provides an environment with the possibility of having a reasonable mixture of high data rates along with a long transmission Range; a combination not facilitated by other dominant wireless protocols of the ISM band. Frequencies selected greater or lower than this range would result in either lower data ranges, or slower data rates. Our selection of the Linvor HC-05 Bluetooth module is based on its widespread utility in numerous Android and Arduino based projects. Following are some features (specifications) of the Linvor HC-05 Bluetooth module followed by its pin-layout and connection setup:

Features	Description
Sensitivity	-80dBm
Output Power (dB)	-4dBm to +6dBm
Flash	8Mb
Operating Voltage	3.1V ~ 4.2V
Current Rating	30mA ~ 4mA (current during communication is 8mA)
Port	HCI
Operating Frequency	2.4 GHz
Bit Error Rate (Avg)	0 (error may occur at transmission link)

Table 1: Summary of Linvor HC-05 Specifications

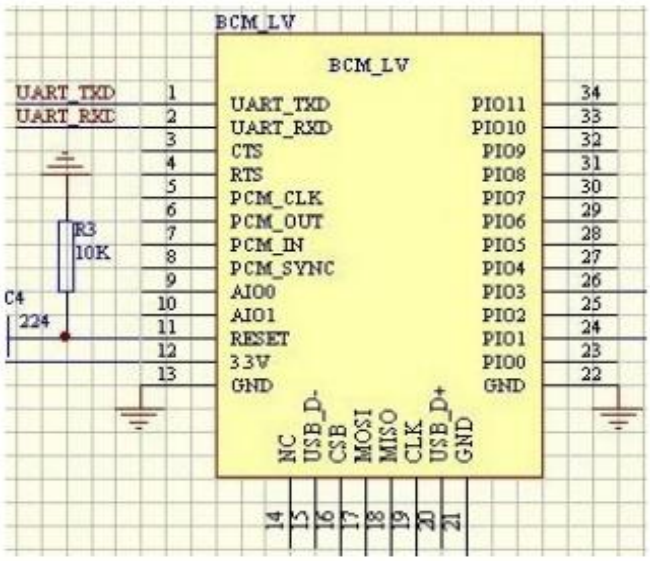


Figure 2: Linvor HC-05 Bluetooth Module Pin-Layout and Connection Setup

Working in close unison with the HC-05, the Teensy 3.1 Controller Module (Teensy) is Encrypting/Decrypting data for Secure Wireless Data Transfer. The Teensy is additionally used to Integrate All the Modules for Synchronized and Efficient Intercommunication between all the modules integrated. For the purpose of our project, we required a processor with enough capacity to run and simulate the AES-128 encryption on a run time basis to support the basic Bluetooth baud rate without hindrance. Bearing this in mind, the Teensy 3.1 embedded with the ARM processor seemed a good choice. This is because of the small size of the processor, as the wireless module being developed has numerous advantages when small in size. The ARM Cortex M family (to which our microprocessor belongs) is a range of scalable, compatible, energy efficient and easy to use processors designed to help developers develop smart embedded applications. The ARM Cortex M4 processor embedded on the Teensy 3.1 is a MK20DX256 – manufactured by Freescale®. Freescale K series controllers (including the one embedded on our Teensy 3.1) are designed for scalable performance, connectivity, communication, and security.

Following are some features (specifications) of the Teensy 3.1 followed by its pin-layout:

Features	Description
Processor	MK20DX256VLH7
Core	Cortex-M4
Rated Speed	72 MHz
Over-clockable	96 MHz

Physical dimensions	36mm*18mm
Flash Memory Cache	256 kB 256 Bytes
RAM	64 kB
EEPROM	2 kB
Direct Memory Access	16 channels
Digital I/O Voltage Output Voltage Input	34 Pins 3.3 Volts 5V Tolerant
Analog Input Converters Usable	21 2 13
Timers	12

Table 2: Summary of Teensy 3.1 Specifications

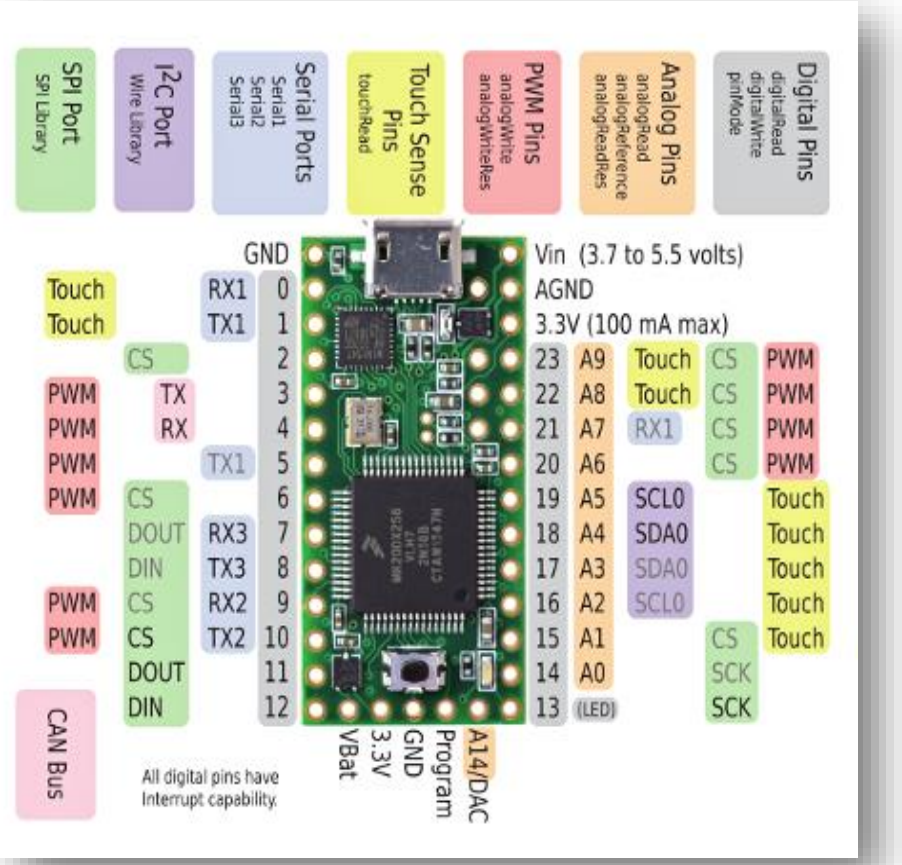


Figure 3: Teensy 3.1 Pin-Layout

The third module of our project is a microSD card. It is serving as an on board data storage unit for our user data. Its miniature size with sufficiently large data storage capacities has been influential in our choice of data storage media. Furthermore, microSD card integration with Arduino is more easily realizable than integrating other storage media.

To access the data content in our wireless USB device on our smart phones, we have designed an android app which sends and receives the contents of the txt file. At the moment as the device which we have designed is just an early prototype, we are currently handling files in txt format only. AES is being implemented on both the microcontroller, as well as on the android application so that the data can be encrypted/decrypted at both the ends.

By virtue of the above mentioned details, we have provided secure communication between two end systems via Bluetooth. We can proudly say that we have designed a fully functional prototype on

which future fyp groups can work to optimize its ability to ultimately send and receive all type of media files, pdf docs, etc.

Chapter 2: Literature Review

The concept of wireless data storage is not an entirely new one; the information age has witnessed and accomplished some remarkable innovations and milestones in data storage and sharing techniques. Congruent to the specifications of our project researchers and technologists have joined forces to develop and present wireless solutions to resolve consumer issues.

One of the first records of the idea can be found in a Masters' (Electrical Engineering) project[2] thesis published in 2004 by Sean Keller. The thesis includes a detailed report on the concept and implementation of a Wireless USB (WUSB) device. His project availed the ISM band to radio-transmit data received from a connected USB v1.1 device using a self-designed protocol.

While working on his WUSB project, Keller investigated an important underlying question; whether the device should follow strict timing requirements of the USB, or should it trick the attached USB and follow the timing standards implemented at the physical level by stretching the protocol to buy time at a transaction level. Keller followed the latter approach for his self-designed protocol.

While designing his own protocol, Keller used the 2.4GHz ISM band – the same frequency band we intend to use, although we intend to employ the standard Bluetooth v2.1 EDR protocol in lieu of any non-standardized self-designed protocol. Advantages of utilizing this deregulated frequency band are reflected in its support for a reasonably high data rate and long transmission range. An in depth justification is discussed in the ***Error! Reference source not found.*** section of the report[3].

Keller's thesis also provided us with an insight on the various classes of transactions available on a USB – all of which will need to be handled wirelessly. Four classes of transactions are ordinarily available on a USB connection: control, interrupt, isochronous, and bulk. Control transactions are required to setup and configure the USB storage device attached to the host. Successful data delivery is ensured through cyclic redundancy checks (CRCs) and retransmission of lost or bad data. Interrupt transfers are used for data polling and their delivery too is ensured through CRCs. Isochronous transfers are used for constant rate error tolerant delivery such as of media files (e.g. audio / video files, etc.) Such transactions have guaranteed bandwidth but no retransmissions take place in case of a failed delivery. The final class is bulk transfers, which is used to deliver large chunks of data on a fixed bandwidth so their delivery is not guaranteed.

[2] Master of Electrical Engineering Program (2004), Cornell University [2].

[3] See also page 5 – ***Error! Reference source not found.***

Working in parallel to Sean Keller, in 2005 the Wireless USB Promoter Group[4] also introduced their own version of the *Certified Wireless USB* protocol – a short-range, high-bandwidth radio transmission protocol. Unlike other models, the WUSB protocol operated at the 3.1-10.6GHz band. Due to its superior bandwidth, WUSB supported greater data rates than other protocols of the time[5]. Unfortunately WUSB faced restrictive legal operation by local regulatory authorities due to its choice in frequency band utilization; the WUSB protocol did not gain much popularity and WUSB products failed to effectively penetrate the market. Witnessing their failure with utilizing the regulated frequency bands has further motivated us to instead opt for the ISM band in our project design.

In contrast to the Wireless USB Promoter Group, some technologists did succeed in producing wireless USB solutions. Of the many technology giants in the flash storage industry, three have launched products of a similar nature to ours – Toshiba®, SanDisk® and Kingston®[6]. They have developed numerous storage devices (flash drives, hard disk drives and memory cards) with embedded wireless modules. In comparison to our project, they chose to implement the IEEE 802.11 (Wi-Fi) standard in their wireless transceiver. An in depth justification of using Bluetooth in lieu of Wi-Fi is discussed in later sections of the report[7].

[4] The WUSB Promoter Group is comprised of Intel®, Microsoft®, Samsung®, HP®, LSI®, and Renesas®.

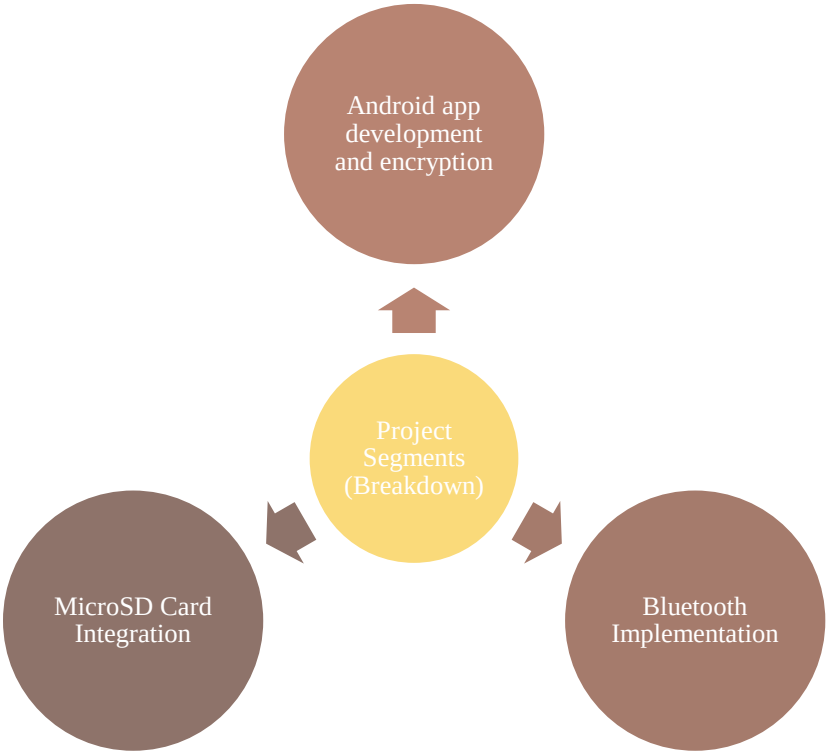
[5] The standard WUSB protocol supported baud rates of up to 480Mbps at distances up to 3 meters.

[6] Commercialized wireless products were launched in early 2014. [6], [7], [8].

[7] See also page 5 – ***Error! Reference source not found.***

Chapter 3: Design and Development

In the beginning of our FYP, our team divided the project into key segments and instead of finishing each segment one after another, we adopted a multitasking approach. Each member was assigned his task along with deadlines so that time could be allocated for the integration of all the key parts with each other. :



3.1 Bluetooth – HC-05 Integration

Our team considered several Bluetooth modules such as BLE112 by BlueGiga®, nRF8001 by Nordic®, CC2540 by Texas Instruments® and the BR-LE4.0-S2A by Blueradios® before finalizing with the Linvor® HC-05. The BLE112 supported Bluetooth v4.0, however the low energy protocol version implemented in the module was not backwards compatible with previous versions of Bluetooth, nor was it compatible with Android phones. The only Android compatible Bluetooth module which did not require an additional development kit was the HC-05 module by Linvor®; hence its selection.

HC-05 runs Bluetooth v2.1 EDR with theoretical data rates of 2-3Mbps. With an 8Mb flash storage and a small size, it definitely is the ideal module for our project.

Figure 4: HC-05 Bluetooth Module

The HC-05 module is configured through serial communication of the AT command set. The HC-05 is programmed[8] using the AT command set. All commands are sent serially to the module. This could be achieved through UART transmission via our controller module. HyperTerminal emulator provides such a computer based interface that enables us to communicate serially through our computer's COM ports.

During this asynchronous transmission of user data or AT commands the sender does not need to send a clock signal to the receiver. Instead, the sender and receiver agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units. When a word is given to the COM port for Asynchronous transmissions (UART), a bit called the *'Start Bit'* is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. The computer's clock and HC-05's internal clock are accurate enough to not have a frequency drift of more than 10% during the transmission of the remaining bits in the word. After the Start Bit, the individual bits of the word of data are sent, with the '*Least Significant Bit' (LSB)* being sent first. Each bit in the transmission is transmitted for exactly the same amount of time as all of the other bits, and the receiver "looks" at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a 1 or a 0 (as in the case of Manchester Encoding). The sender does not know when the receiver has "looked" at the value of the bit. The sender only knows when the clock says to begin transmitting the next bit of the word. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates[9]. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. When the receiver has received all of the bits in the data word it may check for any Parity Bits, and then the receiver looks for a Stop Bit. If the Stop Bit does not appear when it is supposed to, the UART considers the entire word to be garbled and will report a Framing Error to the host processor when the data word is read. The usual cause of a

[8] AT command programming (configuration) is required to alter or reset device name / password / baud rates / master-slave configuration / parity configuration.

[9] This is dependent on the parity configuration we have set through our AT commands.

Framing Error is that the sender and receiver clocks were not running at the same speed, or that the signal was interrupted. If there was no error, and another word is ready for transmission, the Start Bit for the new word can be sent as soon as the Stop Bit for the previous word has been sent.

There are a number of Bluetooth profiles integrated in different Bluetooth modules. The HC-05 Bluetooth module can only operate on the SPP profile. This profile is based on ETSI 07.10 and the RFCOMM protocol. It emulates a serial cable to provide a simple substitute for existing RS-232, including the familiar control signals. It has a maximum payload capacity is 128 bytes. As the HC-05 Bluetooth module doesn't operate on A2DP profile, we can't transfer media files (videos, mp3 etc) through our module. SPP profile is a standard substitute for the RS-232 (Serial communication) protocol; the HC-05 module can only transmit and receive serial data (in ASCII format). The contents of the files transferred and received can be seen on the serial monitor. The files in the SD card can't be shown individually. The contents of each file can only be seen provided the file name is known.

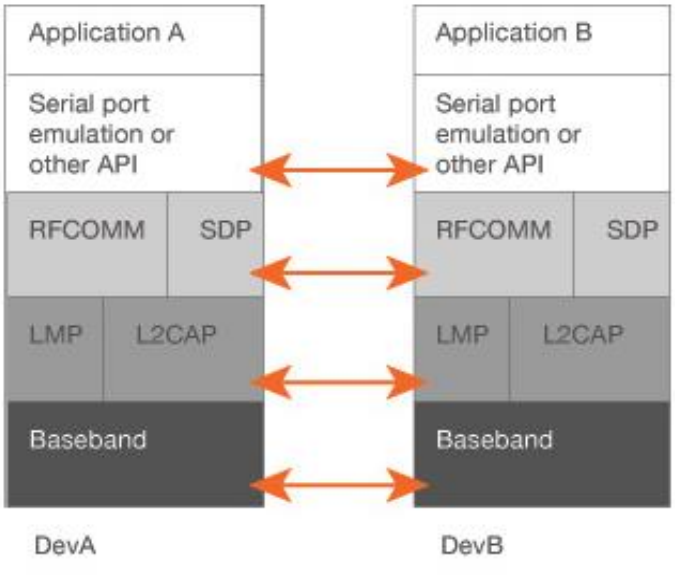


Figure 5 : SPP protocol stack

Above seen is a protocol stack for a SPP protocol. The Baseband, LMP and L2CAP are the OSI layer 1 and 2 Bluetooth protocols. RFCOMM is the Bluetooth adaptation of GSM TS 07.10, providing a transport protocol for serial port emulation. SDP is the Bluetooth Service Discovery Protocol. The applications on both sides are typically legacy applications, able and wanting to communicate over a serial cable (which in this case is emulated). But legacy applications cannot know about Bluetooth procedures for setting up emulated serial cables, which is why they need help from some sort of Bluetooth-aware helper application on both sides.

In this model, there are two devices defined before communication begins. A device that initiates the connection and a device that waits for the other device to take initiative. They are called Initiator & Acceptor respectively.

In our implementation of Bluetooth in our FYP, we have been successful in pairing our module HC-05 with smart phones (Bluetooth enabled). After pairing with the desired device we were also able to send and receive the data (.txt format). Moreover, using the serial monitor and the Android application we were also able to identify that our data that was sent by the controller via HC-05 is correctly and error freely received by the smart phone. Profiles such as A2DP and AVDP were not used due to the complexity of integrating AES. As this is an entry level prototype we have left the task for transfer of media files for the future FYP groups.

3.2 Storage Unit – MicroSD Card Integration

The final segment of our project involves integrating a microSD card with our Teensy to serve as the main user data storage unit. Two factors were influential in our decision to employ a microSD card in lieu of alternative storage media; 1) Notably small physical dimensions, and 2) Presence of online resources and support for Arduino with FAT16 storage formats.

The Teensy is communicating with the microSD card using Serial Peripheral Interface (SPI) communication. We are using Teensyduino v.0.9 add-on with the Arduino IDE to program the Teensy for SPI communication with the microSD card.

To begin communication between our controlling module and SD card module, the SD card module is set as the slave device and the controller as the master device. To begin communication the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports[10]. During each SPI clock cycle, a full duplex data transmission occurs; 1) The master sends a bit on the MOSI line (the slave reads it from that same line), 2) The slave sends a bit on the MISO line (the master reads it from that same line).

Transmission normally involves two shift registers, one as master and the other one as the slave. These registers are connected with each other in a ring. Data moves out as the most significant bit first which lets a new bit in the register in as the least significant bit into the same register. Once all

[10] Since we have not finalized the operating frequency for our Teensy and microSD card, we cannot determine the maximum clock frequency for SPI communication between them.

the values have shifted, the master and slave have exchanged values. The values exchanged are then written onto the memory of the receiving device. If there is more data to exchange, the shift registers are loaded again and thus the process repeats. The serial clock can be either normally high or normally low (clock polarity), and data can be sent on the rising or falling clock edge (clock phase). The four combinations of clock phase and polarity denote the clock mode – numbered 0 to 3. Most SPI chips are designed to work with either mode 0 or mode 3.
The SPI library only supports SPI master mode, where it selects the chip to communicate and the master drives SCK[11] and MISO (master input/slave output) line, and receives on MISO. Our SD card board will be using 4 SPI signals with an additional power and ground line.

In our project we are using 2 GB SD card. Micro SD was a better option in terms of size as the Teensy 3.1 microcontroller itself is in very small size but due the lack of availability micro SD breakout board we were ultimately forced to use a standard SPI based SD breakout board. However, later on we can still use Micro SD once the product is fully developed and ready to be launched into the market.

Following is a figure showing the txt file written on the SD card after its contents were transferred from the smart phone to our Wireless USB device. The same file was then viewed when the SD card was inserted in the card slot of the computer

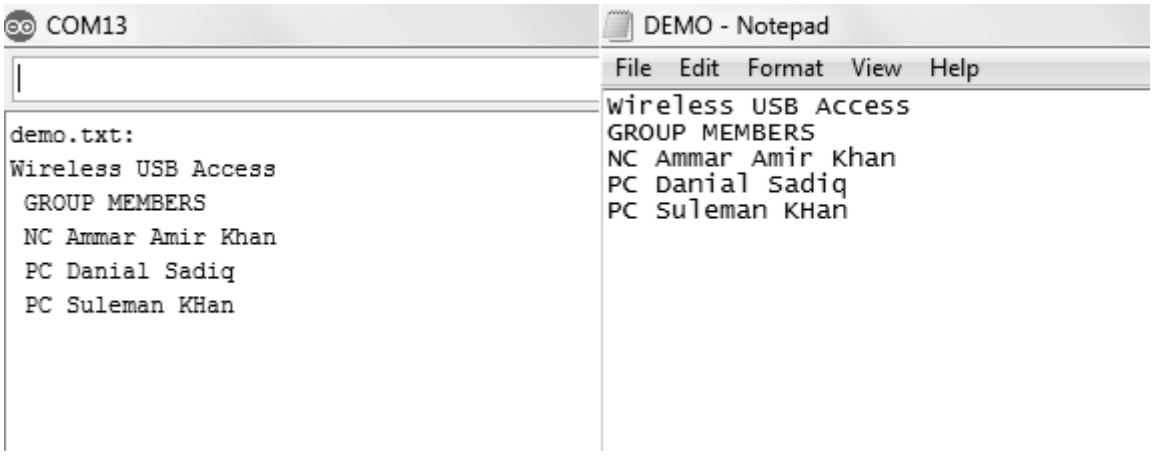


Figure 6: Contents of the SD card shown both on the Wireless USB device and on Computer

[11] With SPI, devices communicate in master/slave mode where the master device initiates the data frame. The SPI bus specifies four logic signals:

- SCLK : Serial Clock (output from master)
- MOSI : Master Output, Slave Input (output from master)
- MISO : Master Input, Slave Output (output from slave)
- SS : Slave Select (active low, output from master)

Error! Reference source not found. outlines the connections required between the Teensy and icroSD card:

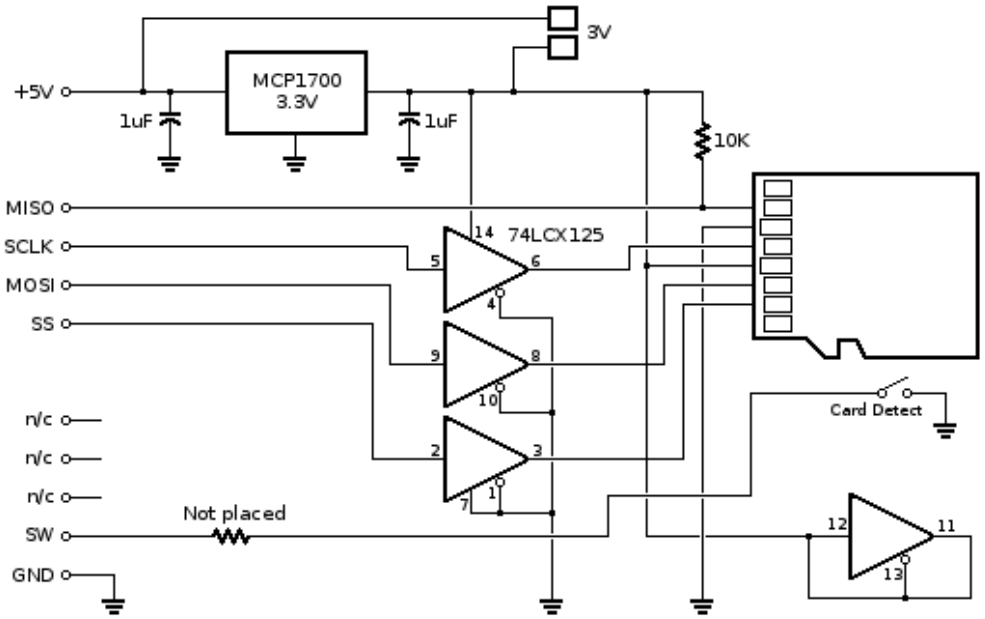


Figure 7: MicroSD Card Connection Setup

3.3 Android Development

As the main objective of our project was to access the user contents in our Wireless USB Storage module on our smart phone, an android application was necessary which paired the smart phone with the wireless device. Furthermore, the application had to also encrypt the data before sending it onto the wireless USB flash device and to decrypt it upon its retrieval. The Wireless USB application was designed on Android studio. The first phase of Android development was to get ourselves acquainted with java environment. Initially the app which was designed only accessed the contents of a txt file

due to the limitations posed on us by the Arduino environment. However, after an extensive phase of research and development we are now able to send and receive txt files as a whole.

Following is the screen shot of the Wireless USB Storage android application

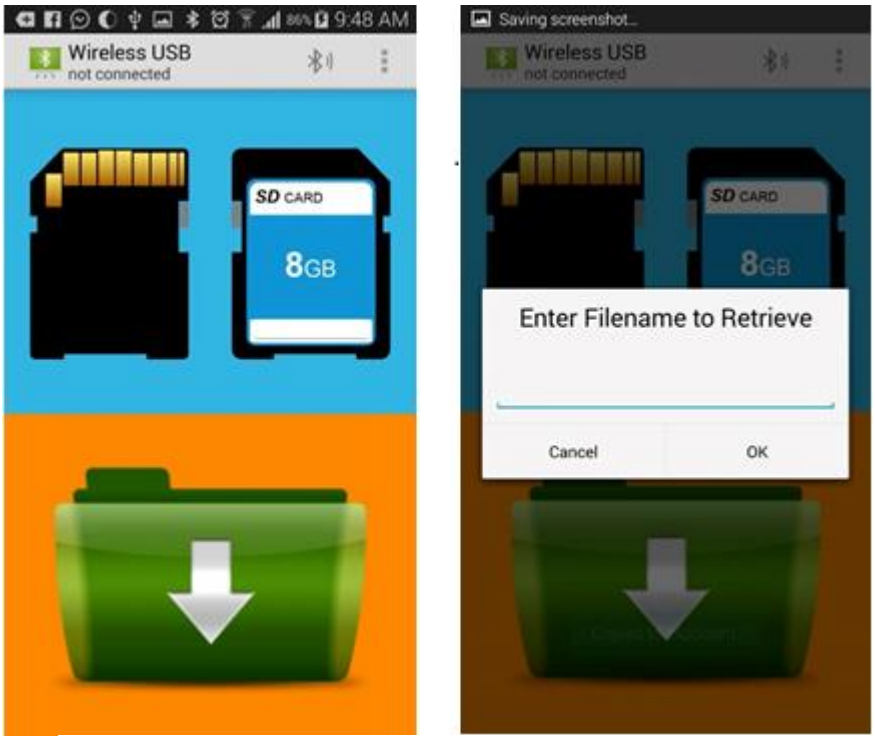


Figure 8: Android Application GUI

3.4 USB Mass Storage

The main purpose of a Wireless USB device should not be only accessing the device wirelessly but also through standard USB peripheral interface so that the computers that are without Bluetooth can also access the user contents stored in the wirless USB module. For this we have configured our microcontroller to run as normal flashdrive upon its connection with any PC through USB port.

The microcontroller after its connection via USB, is viewed as a standard removable flash drive on the computer. This was done by programming teensy's ARM Cortex M4 processor through Atollic IDE. The code developed on Atollic is then burnt onto the teensy via the teensy's boot loader application.

Following is a snapshot of the IDE along with Teensy Boot loader application

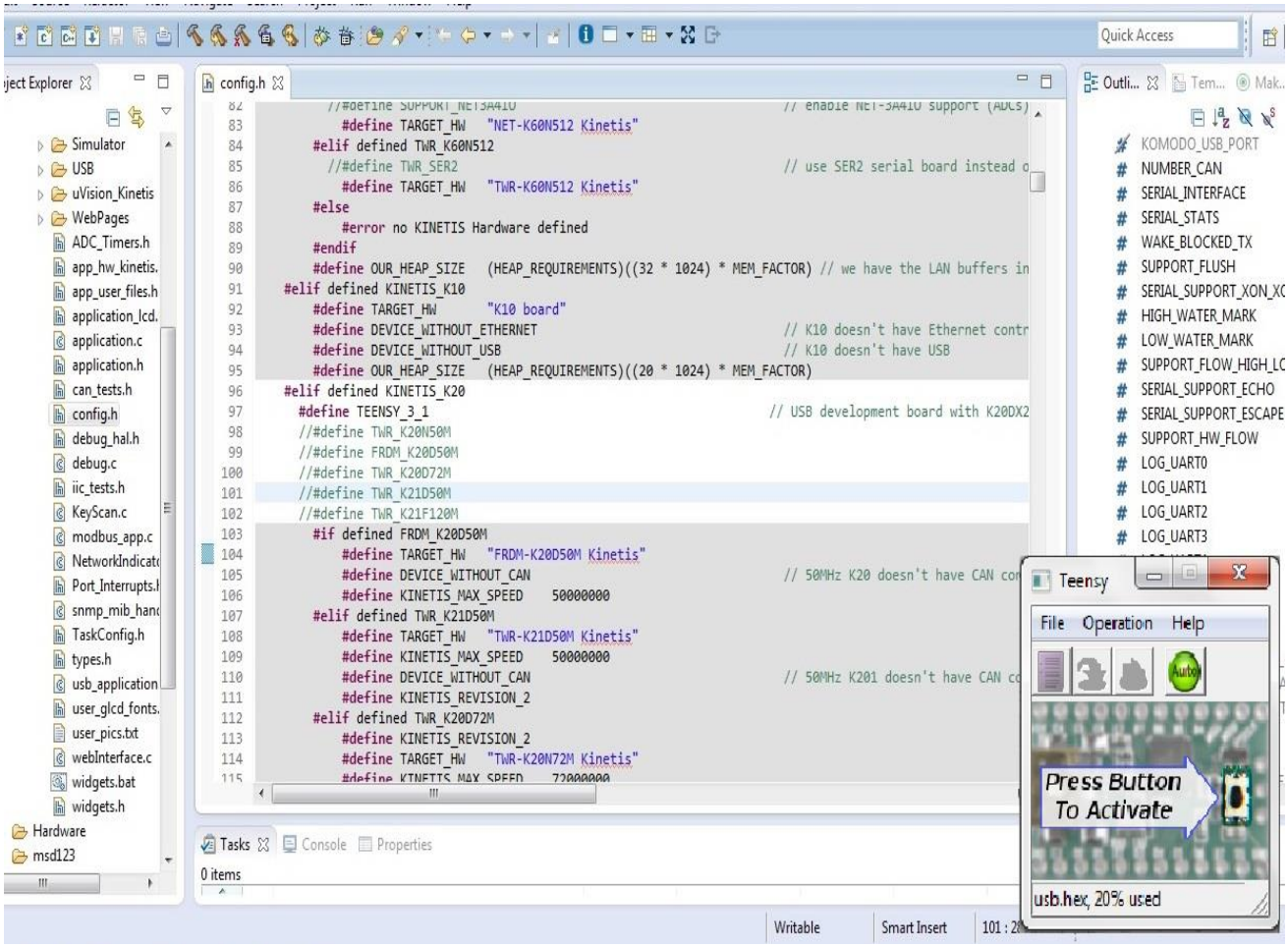


Figure 9: Atollic view along with Teensy Loader Application

3.5 Advanced Encryption Standard (AES-128)

After successfully implementing AES on MATLAB, our team translated the code into Arduino environment. The AES segment can further be divided into two parts; 1) AES on Teensy, 2) AES on Android. Our selection of the AES encryption standard is based on its universal acceptance in cryptographic modules worldwide due to its robustness against hackers and interceptors.

Previously, when implementing the algorithm in MATLAB we were using the dynamic keys (Add Round Key function) for encryption and decryption, but there was a high probability that the keys generated by the transmitter and receiver will not be the same and that could disrupt the communication. So, we have left the concept of dynamic keys and shifted to the concept of static keys. These keys have been taken from our MATLAB implementation. Once defined, the keys aren't changed, however, they are only shared between the user (any Bluetooth enabled device) and the

admin (controller module). These keys have been defined clearly in our implementation as we shall see in the next section. The transformation functions have also been defined in our implementation. In the next segment we discuss each of the functions separately.

3.5.1 Algorithm Breakdown

The algorithm must implement symmetric key cryptography as a block cipher and support minimum block sizes of 128 bits with key sizes of 128, 192, or 256 bits[12]. We had three main design goals in developing and implementing the code for AES; 1) Design simplicity, 2) Resistance against known attacks, and 3) Speed and efficiency with code compactness for both our intended platforms.

To increase the efficiency and speed of the algorithm; the function of Key expansion is not needed. As the keys are already pre-defined for each round we can skip the function. This will not only smooth the process of encryption but it will take less time to encrypt and decrypt the data.

AES_AMMAR_mixcol

```
unsigned int key2[4][4] =
{
  {0x23,0x60,0x25,0x6D},
  {0x3B,0x3B,0x3B,0x3B},
  {0x21,0x04,0x72,0x3B},
  {0x52,0x52,0x52,0x52}
};

unsigned int key3[4][4] =
{
  {0xC3,0xA3,0x86,0xEB},
  {0xD9,0xE2,0xD9,0xE2},
  {0x21,0x25,0x57,0x6C},
  {0x6E,0x3C,0x6E,0x3C}
};
```

Figure 10 : Static Keys Defined

In the figure above we see that the static keys are of the order 4x4. In total the key size is 128 bits.

[12] We have already implemented AES with a key size of 128 bits. We are trying to now implement AES with key size of 256 bits on our controller (due to its added robustness).

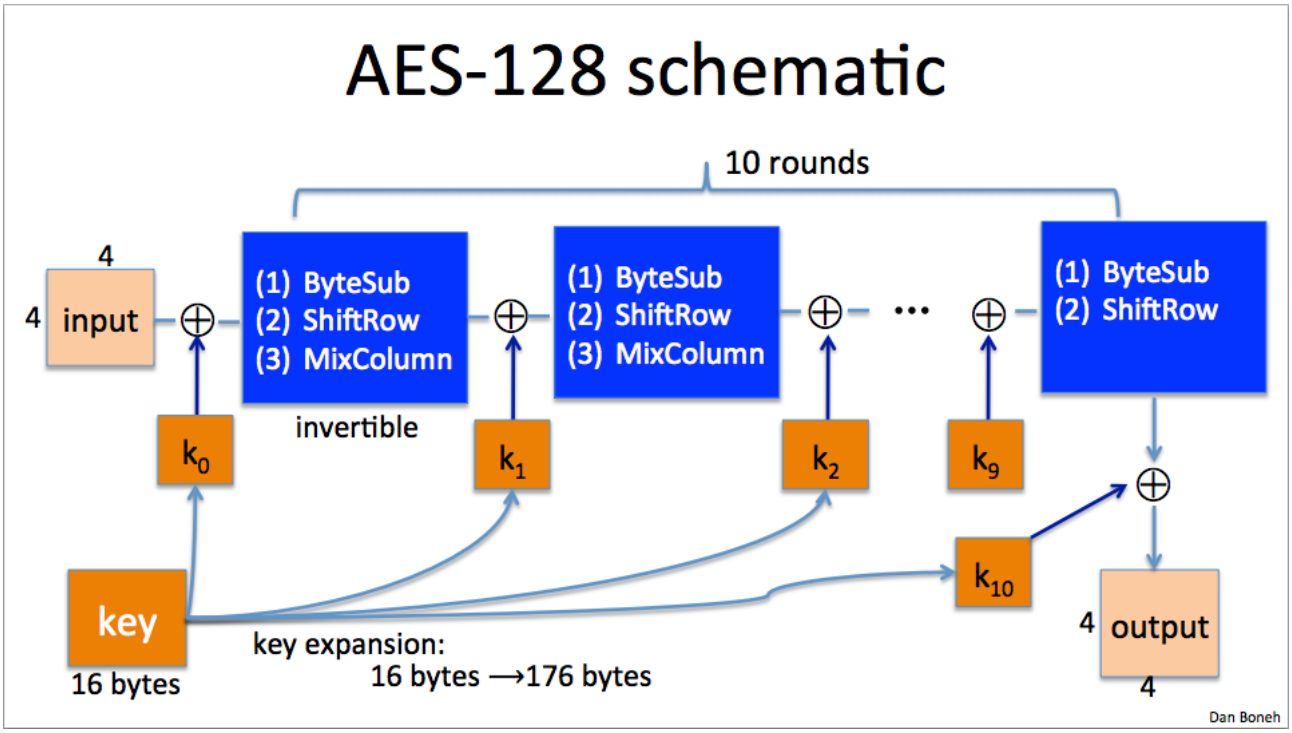


Figure 11 : AES-128 Schematic

3.5.1.1 Add Round Key Operation

The first operation on the state matrix itself is the Add Round Key process. The Add Round Key process is essentially an element-wise exclusive-or operation (XOR) between the state matrix and the round key. This is achieved by XOR-ing the state and the round key generated by the key expansion function above. As stated before, these keys are pre-defined as Static Keys, which are called before XORing with the state matrix.

The Add Round Key process is invertible by performing the operation twice, which is why the Add Round Key operation is used in decryption as well. An initial Add Round Key operation is performed on the state matrix directly after key expansion and before it goes through the main encryption body. In the main encryption body, an Add Round Key operation is performed at the end of every round, and the key involved in the operation is different for every round. The algorithm adds the eleventh round key after the main body of the encryption procedure in the final step – thus completing the entire encryption process.

At the end of each round after the three operations of Sub-bytes, Shift rows and Mix column Add round key function is applied. Each round has its own key is defined. That key is XOR'd with the present state of the input column wise.

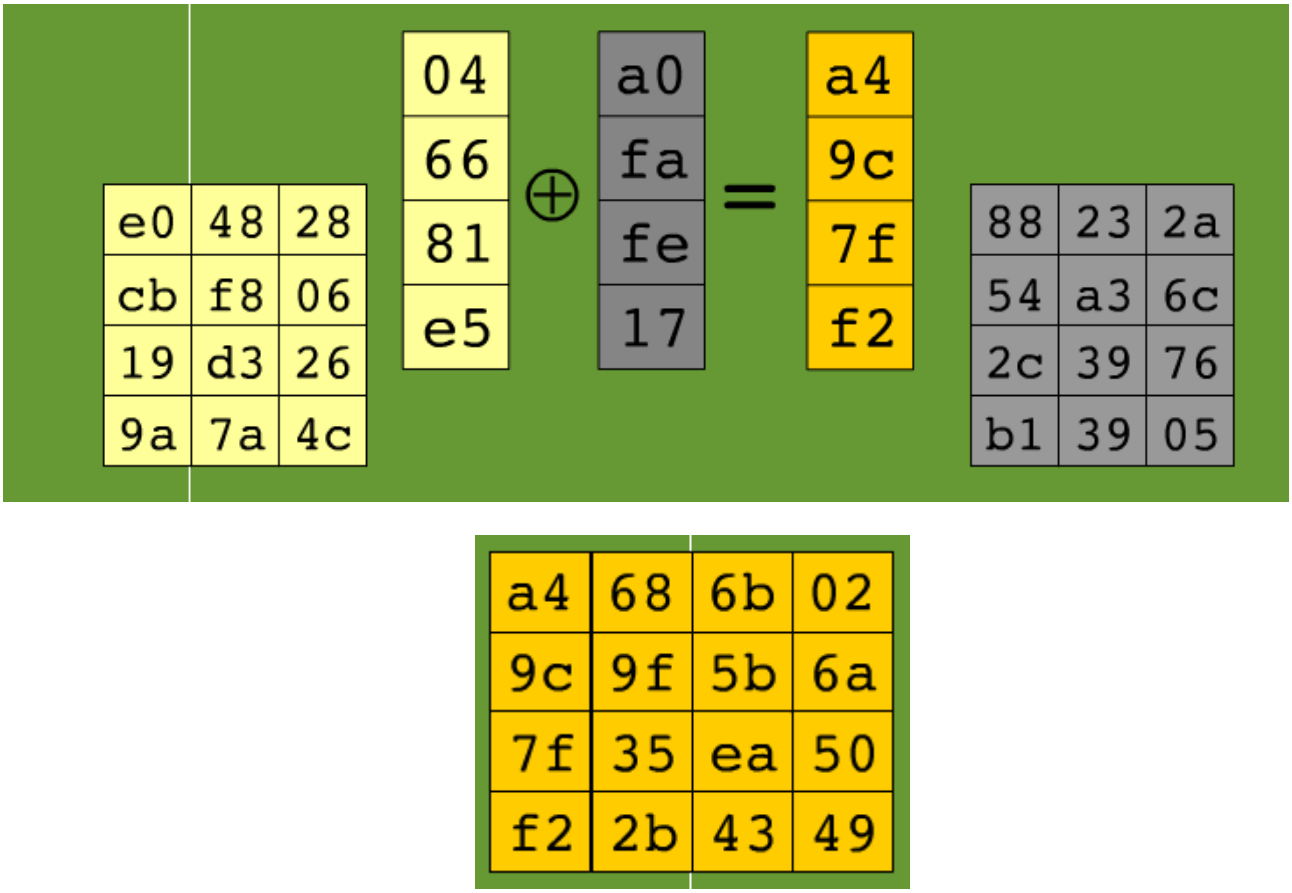


Figure 12 : Add Round Key Function & the Final State after the Operation

3.5.1.2 Sub Bytes Operation

The sub-bytes operation is the first operation performed on the state matrix on every round of encryption. It is also performed directly after the last round in the final part of the algorithm. The sub-byte operation is essentially a table lookup that uses the Rijndael S-box. During decryption, an inverse S-Box is also generated and called when required.

```
unsigned int sbox[16][16] =
{
  {0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76},
  {0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0},
  {0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15},
  {0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75},
  {0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84},
  {0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF},
  {0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8},
  {0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2},
  {0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73},
  {0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB},
  {0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x9A,0x95,0xE4,0x79},
  {0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08},
  {0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A},
  {0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E},
  {0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF},
  {0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16}
};
```

Figure 13: S-box used for Encryption

Using this s-box is quiet simple. The LSB of the input in searched in the rows and the MSB of the input searched through the columns and the value is replaced at the intersection.

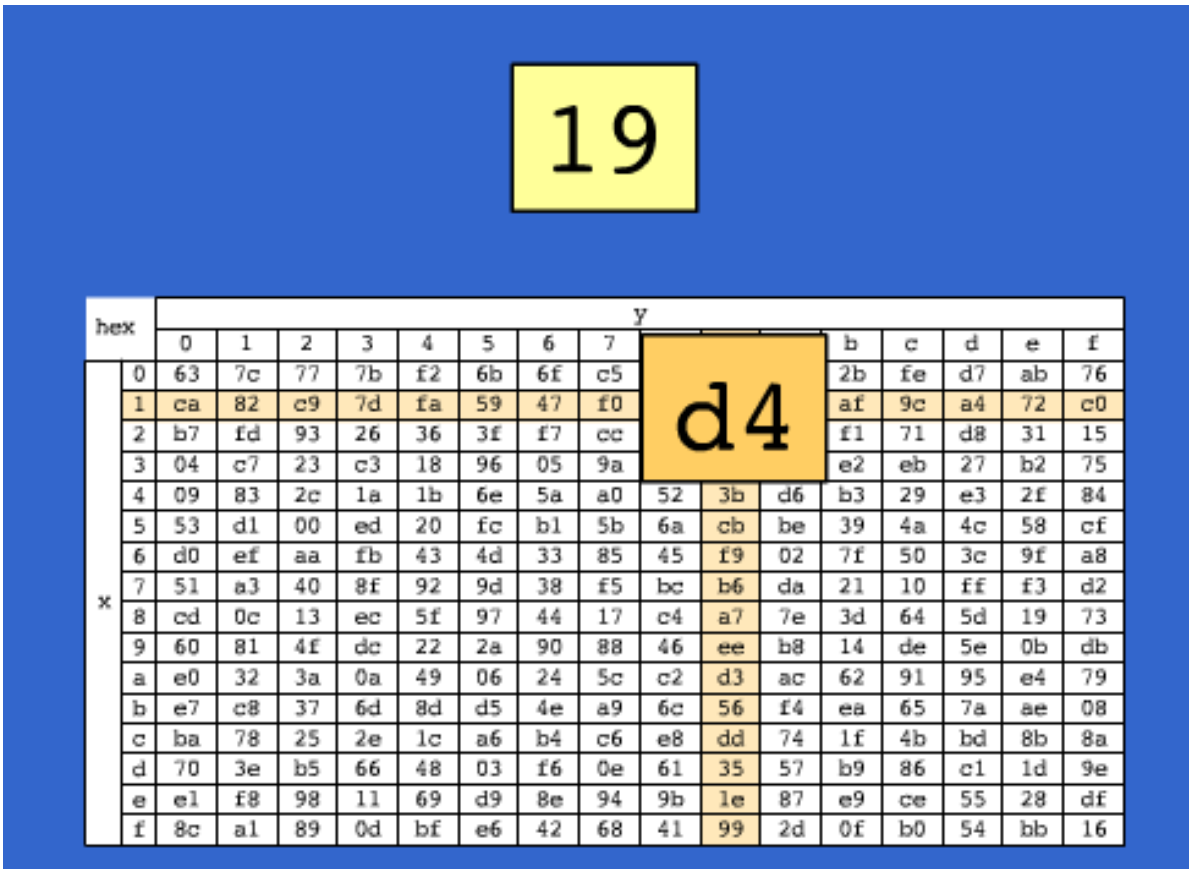


Figure 14 : Replacing the value from s-box

```
unsigned int invsbox[16][16] =
{
  {0x52,0x09,0x6A,0xD5,0x30,0x36,0xA5,0x38,0xBF,0x40,0xA3,0x9E,0x81,0xF3,0xD7,0xFB},
  {0x7C,0xE3,0x39,0x82,0x9B,0x2F,0xFF,0x87,0x34,0x8E,0x43,0x44,0xC4,0xDE,0xE9,0xCB},
  {0x54,0x7B,0x94,0x32,0xA6,0xC2,0x23,0x3D,0xEE,0x4C,0x95,0x0B,0x42,0xFA,0xC3,0x4E},
  {0x08,0x2E,0xA1,0x66,0x28,0xD9,0x24,0xB2,0x76,0x5B,0xA2,0x49,0x6D,0x8B,0xD1,0x25},
  {0x72,0xF8,0xF6,0x64,0x86,0x68,0x98,0x16,0xD4,0xA4,0x5C,0xCC,0x5D,0x65,0xB6,0x92},
  {0x6C,0x70,0x48,0x50,0xFD,0xED,0xB9,0xDA,0x5E,0x15,0x46,0x57,0xA7,0x8D,0x9D,0x84},
  {0x90,0xD8,0xAB,0x00,0x8C,0xBC,0xD3,0x0A,0xF7,0xE4,0x58,0x05,0xB8,0xB3,0x45,0x06},
  {0xD0,0x2C,0x1E,0x8F,0xCA,0x3F,0x0F,0x02,0xC1,0xAF,0xBD,0x03,0x01,0x13,0x8A,0x6B},
  {0x3A,0x91,0x11,0x41,0x4F,0x67,0xDC,0xEA,0x97,0xF2,0xCF,0xCE,0xF0,0xB4,0xE6,0x73},
  {0x96,0xAC,0x74,0x22,0xE7,0xAD,0x35,0x85,0xE2,0xF9,0x37,0xE8,0x1C,0x75,0xDF,0x6E},
  {0x47,0xF1,0x1A,0x7A,0x1D,0x29,0xC5,0x89,0x6F,0xB7,0x62,0x0E,0xAA,0x18,0xBE,0x1B},
  {0xFC,0x56,0x3E,0x4B,0xC6,0xD2,0x79,0x20,0x9A,0xDB,0xC0,0xFE,0x78,0xCD,0x5A,0xF4},
  {0x1F,0xDD,0xA8,0x33,0x88,0x07,0xC7,0x31,0xB1,0x12,0x10,0x59,0x27,0x80,0xEC,0x5F},
  {0x60,0x51,0x7F,0xA9,0x19,0xB5,0x4A,0x0D,0x2D,0xE5,0x7A,0x9F,0x93,0xC9,0x9C,0xEF},
  {0xA0,0xE0,0x3B,0x4D,0xAE,0x2A,0xF5,0xB0,0xC8,0xEB,0xBB,0x3C,0x83,0x53,0x99,0x61},
  {0x17,0x2B,0x04,0x7E,0xBA,0x77,0xD6,0x26,0xE1,0x69,0x14,0x63,0x55,0x21,0x0C,0x7D}
};
```

Figure 15: Inverse S-box used for Decryption

3.5.1.3 Shift Row Operation

The shift rows operation takes every row of the state matrix and shifts it towards left. The first row remains un-changed; the second row is shifted left by one, the third row by two, and the fourth row by three. The shift is circular in that elements at the left side that are ‘cut off’ by the shift are replaced in empty spaces on the right side of the row.

We have simply used the rotation function to implement the function in the Teensy.

d4	e0	b8	1e	d4	e0	b8	1e
27	bf	b4	41	bf	b4	41	27
11	98	5d	52	5d	52	11	98
ae	f1	e5	30	30	ae	f1	e5

Figure 16 : Before & After the Shift Row Operation

3.5.1.4 Mix Column Operation

The mix columns operation is essentially a matrix multiplication between the state matrix and a given matrix column mixer matrix. Normal scalar addition and multiplication operators are not involved in the multiplication of the columns mixer matrix and the state matrix. However they are different in the sense that addition is an exclusive-or (XOR) operation and multiplication is Galois field multiplication in the number field.

There is Rijndael constant matrix which is predefined for the Mix Column operation.

```
unsigned int mixer[4][4] =
{
   {0x02,0x03,0x01,0x01},
   {0x01,0x02,0x03,0x01},
   {0x01,0x01,0x02,0x03},
   {0x03,0x01,0x01,0x02}
};
```

Figure 17 : Rijndeal Constant

When multiplied by 0x01 the entries are simply XOR'd but the multiplication by 0x02 and 0x03 aren't simple. When multiplied by 0x02 the input matrix is shifted twice towards left like the Shift Row operation. Then all the results of the row are XOR'd to get the final state.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb. \qquad (5.6)$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Figure 18 : Mix Column Equations

Chapter 4: Project Analysis and Evaluation

Now, as we have successfully paired our device with an android smartphone, we can send and receive .txt files of any size. However there is still a large room for development. Media files such bitmap, jpeg can still be sent or received but due to lack of error correction techniques these files often get corrupt during transmission. By the use of error correction schemes such as repetition codes, parity bits, checksums, cyclic redundancy checks etc., we can ensure that the data transmitted/received is error free. This task, therefore we leave for the future FYP groups.

The GUI that we designed for the android app is very simple and easy to use. Simple commands of sending and receiving can be sent to the Wireless USB device. By accessing file manager through the app, the file which we want to send for storage can be selected. After it's encrypted via AES 128, it is sent to the Wireless USB device and stored on to the SD card. To retrieve the file, again a command

of prescribed numeric character is given upon which the file name of the required file is asked for retrieval. The file is thus retrieved and stored to the Bluetooth receive folder of the smartphone.

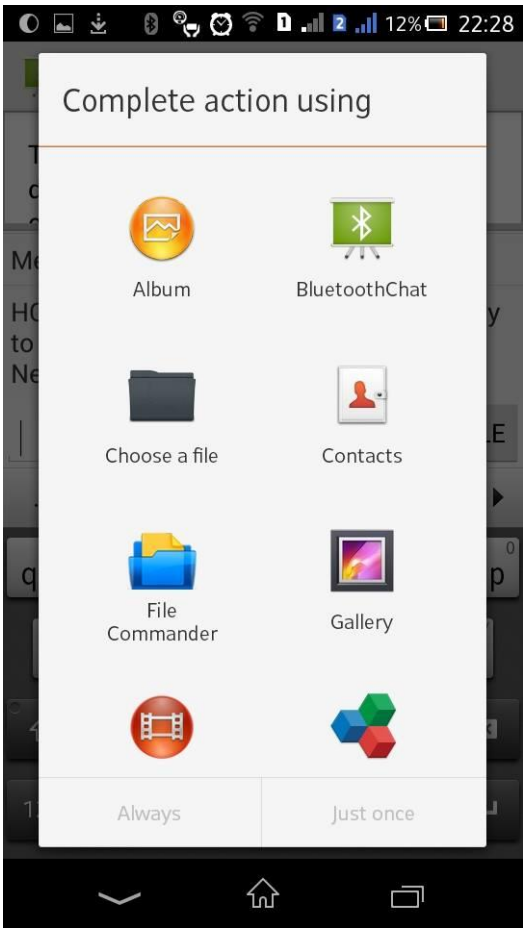


Figure 20: App showing Sending mode

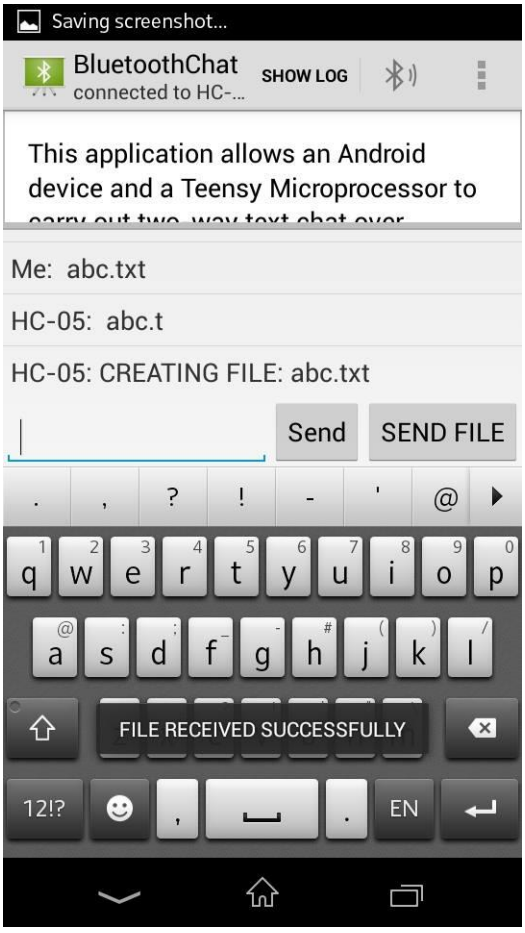


Figure 19: App showing File Receiving mode

AES 128 encryption technique is one of the most secure encryption techniques used today. It can be implemented on processors as small as 8 bit ones. With some performance trade off, it takes minimum RAM with minimum clock cycles. As compared to 256 bit AES, the 128 bit one takes less rounds and thus less memory however, it's vulnerable to attacks as compare to AES 256. As the Bluetooth

has a very limited range, attacking cyphers on our USB will really be a challenging task. Following is a demonstration of AES 128 on Arduino.

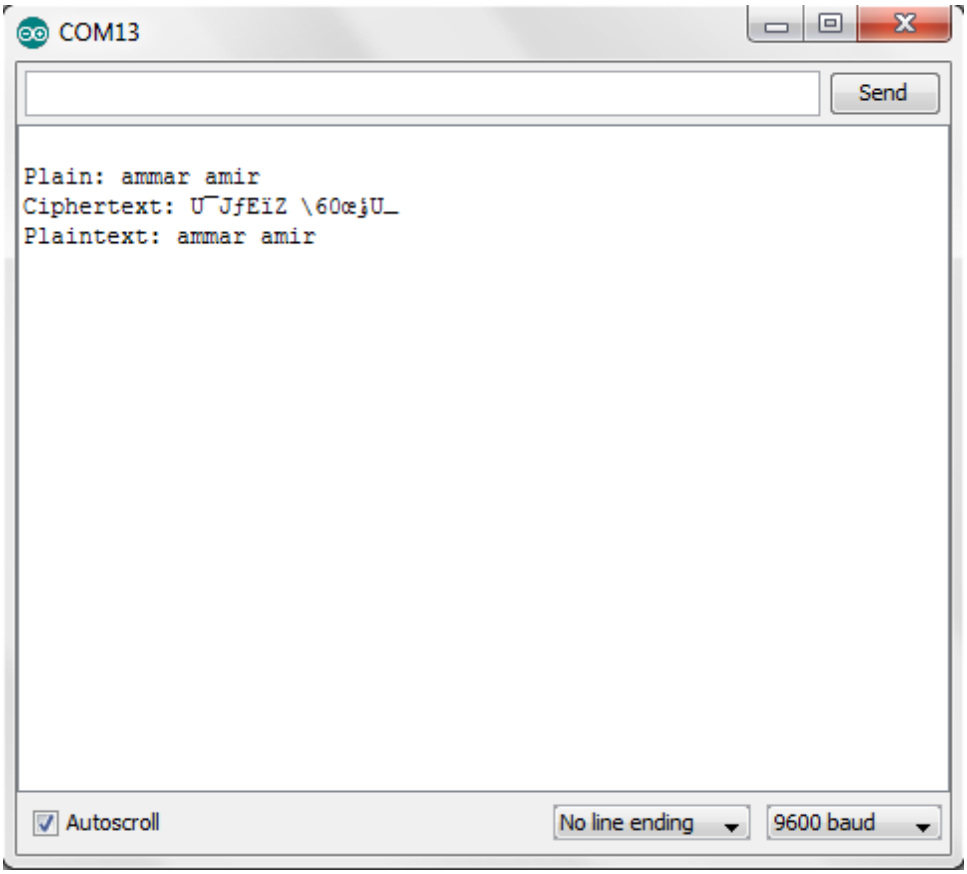


Figure 21: AES 128 implementation on Arduino

During the course of our FYP, size of the device is another factor which we have tried to work on. Teensy microcontroller, apart from its set of abilities, is really small in size. However, as the modern day USB flash drives have decreased in size to the extent that they are now smaller than a thumb size and thinner to a few millimeters, our device is still large and heavy. However, through R&D in the future, the size can surely be decreased.

Chapter 5: Recommendations for Future Work

For the future, Bluetooth modules 4.0 and higher can be used for a faster data delivery. Modules such as of bluegiga and blueradios can be integrated with our existing controlling module. As these modules work on AVDP and A2DP profiles, the device will thus be able to transfer all type media files along with doc, pdf files etc.

Furthermore, higher encryption standards such as AES 192, 256 can implemented for enhanced level of security for the device. The size of the product can also be worked upon; it can be reduced using different new technologies & powerful processors can be used to increase the processing power even more.

Wi-Fi is another option which can be integrated. Doing this will increase the number of ways through which user can access his data on the move. This option however, will require extremely high levels of security as the cloud storages are more prone to hacking than any other form of data storages.

Chapter 6: Conclusion

After a yearlong work on this project, we as a team achieved certain objectives, there were certain limitations and there were certain achievements. There is no better way to conclude this project but to take on the journey of what, how & why it was developed.

6.1 Overview:

Since its inception, Universal Serial Bus (USB) has become the standard peripheral interface in personal computers. However with the evolution of smart technology and competent communications' networks, consumer trends have now shifted towards wireless data storage. At the moment, most USB flash drives have no inbuilt wireless facility. By designing and developing an interface between Bluetooth and USB protocols, we will alleviate contemporary USB-based storage media of their pertinent electrical connections.

In a nutshell, this project allows one to transmit & receive data wirelessly between two Bluetooth enabled devices. This transmission & receiving of data can be secured using different Information Security techniques and here, we have used the Advanced Encryption Standard (AES) to serve the purpose.

6.2 Objectives Achieved:

Different Academic & project objectives were set & achieved during the course of this project. As mentioned before the academic objectives achieved are discussed below.

6.2.1 Academic Objectives

By implementing the ARM processor techniques we were able to achieve the embedded system design objective. The implementation of Advanced Encryption Standard allowed us to explore the Information Security sector in detail as well. Moreover, the wireless communication through the ISM band enabled us to exploit the field of wireless communications. We were able to identify the pros & cons of different wireless techniques and implement the most suitable one i.e. Bluetooth at 2.4 GHz. Additionally, by designing & implementation of the android application through Android Studio we were able to exploit the Android software and learned how to develop basic GUI interfaces as well as technical aspects of the software.

6.2.2 Project Objectives

We have contrived a possible prototype for wireless communication via Bluetooth channel over USB peripherals. By incorporating the AES in our project, we have been able to achieve secure wireless transfer of data via Bluetooth. By choosing the right hardware we have developed a device that is portable and can be carried like a normal USB. With further research and development, this prototype can be eventually transformed into a fully functional product that can be launched into the market.

6.3 Achievements

Following are the most prominent achievements during the course of project.

- Wireless Transfer of data via Bluetooth.
- Interfacing the Bluetooth module (hard-wired) to the controller module Teensy 3.1.
- Interfacing the microSD card (hard-wired) with the controller module Teensy 3.1.
- Designing & Implementing the Android application to send & receive data via Bluetooth channel.
- Using the ARM Cortex M4 processor on the controller module Teensy 3.1, the microSD card module is converted to Mass Storage Device (MSD) that can be plugged in and out like a Universal Serial Bus. In short the microSD card module is used as plug & play device with any computer as well.
- Dual task of the microSD card modules have been achieved; the Wireless Storage & the Mass Storage Device (MSD).

6.4 Limitations

Where there are a lot of advantages & applications of the project, there are definitely some limitations also.

Currently, the project can only transmit the ".txt" extension files. Moreover the baud rates, though they are optimized, they are a little bit slow compared to other technologies offered in the market. As mentioned before, Bluetooth modules with profiles like A2DP & AVDP can be used to transfer the media files (audio, video & high resolution images) & the baud rates can also be increased with efficient data handling techniques.

APPENDIX – 1

PROJECT SYNOPSIS

Extended Title:

Wireless USB Access – An Intermediate Between USB and RF Technology

Brief Description of The Project / Thesis with Salient Specs:

We will design a two part data transceiver module equipped with two communications' links; (1) A wireless module (probably Bluetooth due to the short-ranged data transfer functionality of our project); and (2) A USB module (to directly connect with a USB device for guided connection). Essentially, it will be an intermediary interface to connect a simple USB device with another wireless or USB enabled device.

Scope of Work :

Wireless data accessibility will primarily ease and facilitate data transfer between devices. This ease of access will be useful for personal / sensitive data transfers, as well as special cases where short ranged remote data collection is required.

Academic Objectives :

Our academic objectives will be to design and contrive a useful product by virtue of our theoretical knowledge base. In the process, we will learn about:

- RF Technology
- Controller Interfacing
- Digital Communication
- Practical Circuit Designing

Application / End Goal Objectives :

To communicate between an end system and a USB device wirelessly, as well as through a USB port.

Previous Work Done on The Subject :

WUSB technology was developed by the WUSB Promoter Group previously; although it did not gain much popularity. Also,
A project of the sort was done in Cornell University in 2004:

http://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2003to2004/sjk26/WUSB.pdf

http://www.usb.org/developers/wusb/

Material Resources Required :

USB Storage Device, Wireless Module (probably Bluetooth), Controlling Hardware.

No of Students Required : 3

Special Skills Required :

Technical skills required for hardware integration; programming for controlling hardware, Bluetooth module integration (digital communication theory), and PCB design and implementation.

APPENDIX – 2

TIMELINE OF THE PROJECT

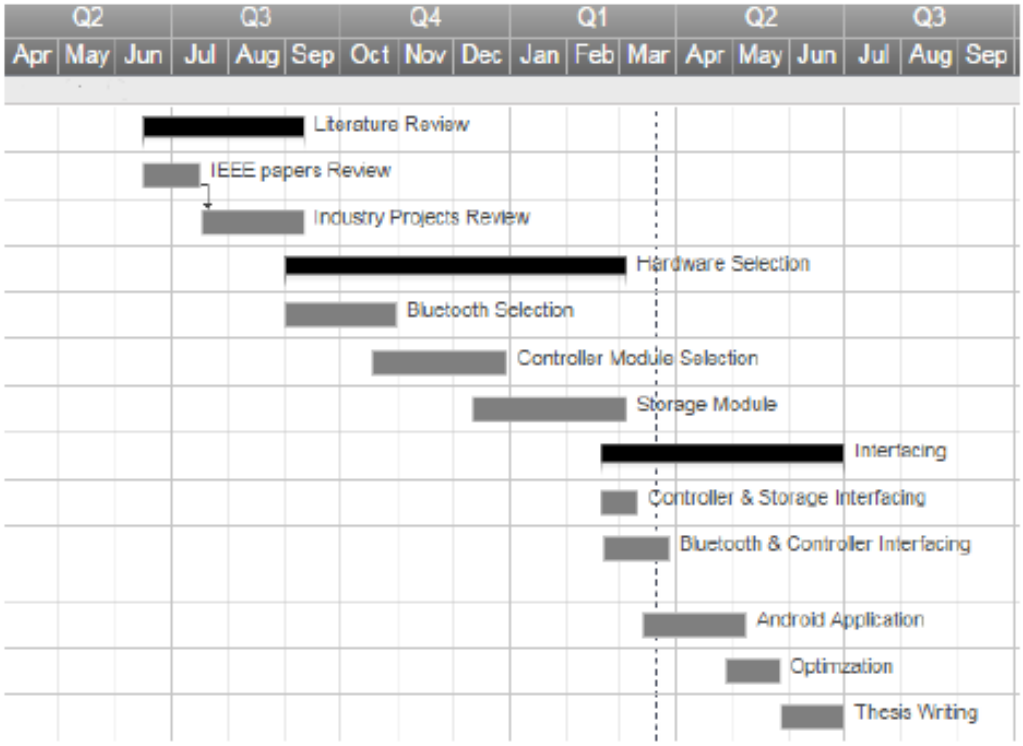


Figure 22 : Timeline

APPENDIX – 3

PROJECT COST BREAKDOWN

Component	Cost (Pk. Rupees)
Teensy 3.1 Controller Modules (x2)	9000
HC-05 Bluetooth Module (x2)	2000
SD Card (Kingston 2GB) (x2)	500
micro SD Card Shield	2000
Printing	5000
Travelling	5000
Miscellaneous	1500
Total	**24000**

Table 3: Cost Breakdown

APPENDIX – 4

FINAL PCB DESIGN ON THE CIRCUIT

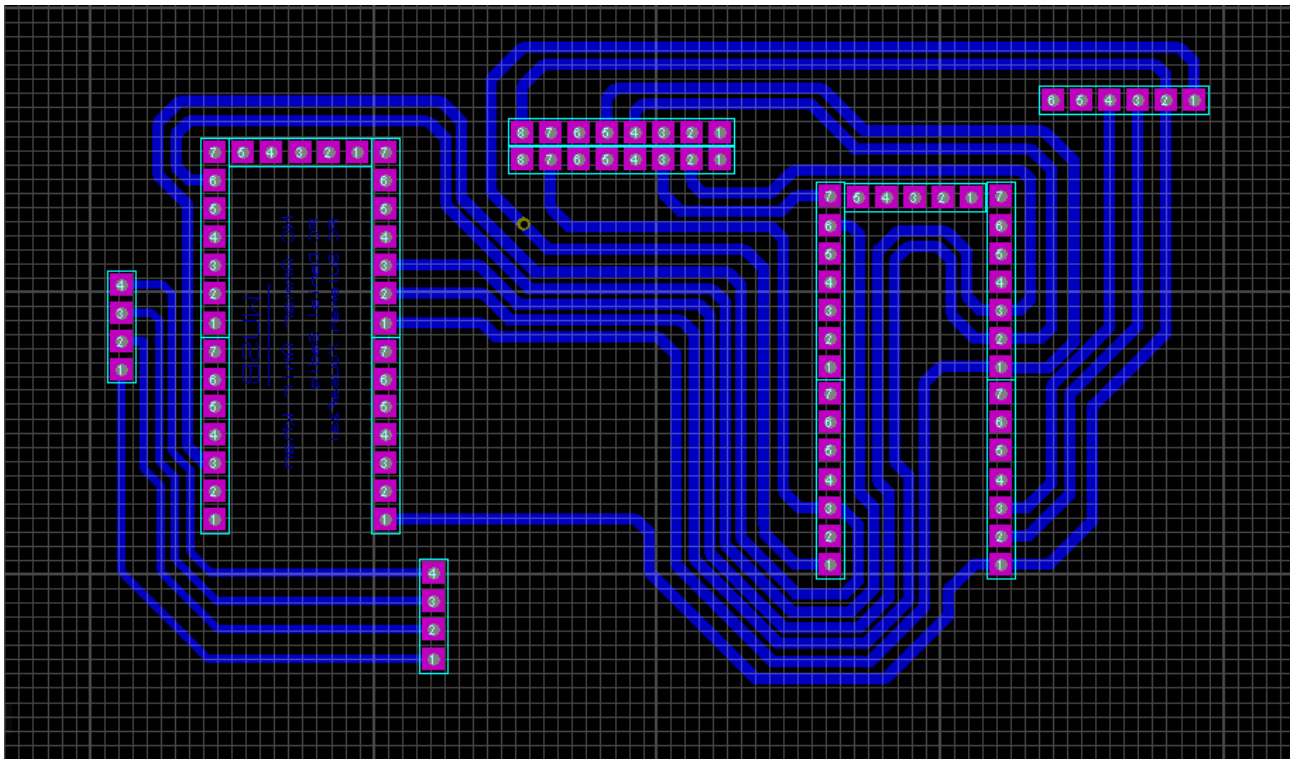


Figure 23 : PCB (ARES) Design

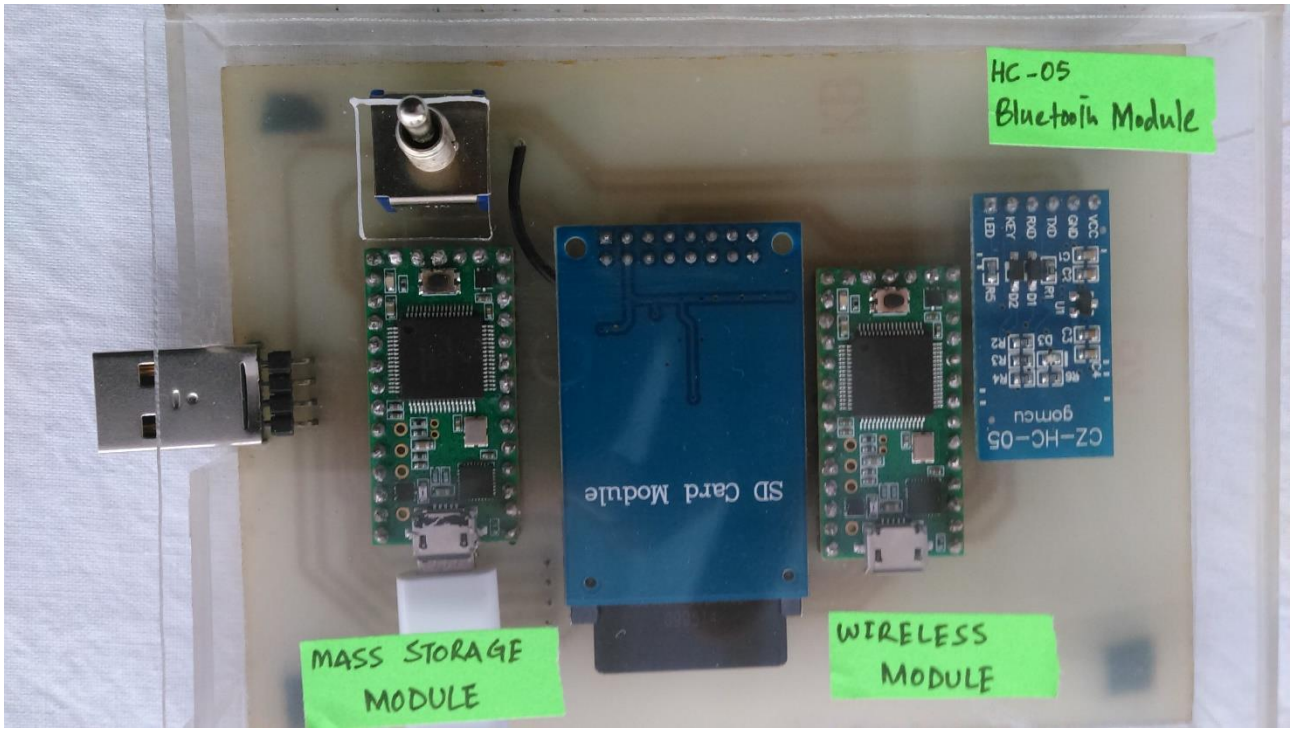


Figure 24 : Final Product Implementation

APPENDIX – 5

Wireless USB Implementation on Arduino

```
#include <SD.h>
#include<SPI.h>

File myFile;
int ind = 0;

void setup()
{
 // Open serial communications and wait for
port to open:
  Serial.begin(9600);
  Serial3.begin(1382400);

  File myFile;

  delay(1000);

  Serial.print("Initializing SD card...");



  SPI.setMOSI(7);
  SPI.setMISO(8);
  SPI.setSCK(14);
  pinMode(6, OUTPUT);

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  //receiveFile();
  //sendFile();

}

void loop()
{
  Serial3.write("    ");


  int choice;
```

```
 while ((Serial3.available() <= 0)); // If there
is no data Stay Here !!!

  while((Serial3.available() > 0)) // Checks for
a character in the serial monitor
{
choice = Serial3.parseInt();
Serial3.write(choice);
break;
}

    if(choice == 1) {Serial3.write("Arduino:
Ready to receive...");receiveFile();}
    else if(choice == 2)
{Serial3.write("Arduino: Ready to
send...");sendFile();}
    else {Serial3.write("If/Else Loop Error!");}
}



void receiveFile()
{
  Serial3.println("Ready to Receive... / ");
  Serial3.flush();
  String fileName_str;
  String readString;

  Serial.print("New file name: ");
  Serial3.write("New file name: ");
  Serial3.flush();
  while (Serial3.available() == 0) {}
  fileName_str = Serial3.readString();

  if (readString.length() > 0)
  {
    readString = "";
  }
  fileName_str.trim();

  Serial3.println("Creating file...");
  Serial3.flush();

  char fileName[fileName_str.length()];
  fileName_str.toCharArray(fileName,
fileName_str.length()+1);
```

```
  myFile = SD.open(fileName,
FILE_WRITE);

  Serial3.println("Send File: ");
  Serial3.flush();

  byte buffer[500]; // Setting buffer of 500
Bytes
  //delay(7000);
 while(Serial3.available()<=0);
  while(Serial3.available()>0)
{
  Serial3.println("Inside the buffer loop: ");
int x = Serial3.readBytes(buffer , 10000);
myFile.write(buffer,x);
}


  if(SD.exists(fileName))
  {
    Serial3.println("File created...");
    Serial3.flush();
    //fn = 0;
    //main();
  }


 else
  {Serial3.println("FILE NOT Created");
   Serial3.flush();}



  myFile.close();
}




void sendFile()
{
  Serial3.println("SEND FUNCTION / ");
  Serial3.flush();
  String fileName_str;
  String readString;


  Serial3.println("Starting receiving mode...");
  Serial3.flush();
  //Serial3.print("SYS_MSG_RCV");
  //Serial3.flush();
  Serial3.println("Retrieve file (name): ");
  Serial3.flush();
  //Serial3.write("Retrieve file (name): ");
  //delay(5000);

while(Serial3.available() <=0 );

  while (Serial3.available() > 0)
  {fileName_str = Serial3.readString();}



  fileName_str.trim();

  char fileName[fileName_str.length()];
  fileName_str.toCharArray(fileName,
fileName_str.length()+1);

  Serial3.println(fileName);
  Serial3.flush();
  myFile = SD.open(fileName);

  Serial3.print("SYS_MSG_RCV");
  Serial3.flush();
  if(!SD.exists(fileName))
  {Serial3.println("File not found!");
   Serial3.flush();}

  else
  {
    delay(2000);
  Serial3.println(fileName);
  Serial3.flush();
  delay(500);
  Serial3.print(myFile.size());
  Serial3.flush();
  delay(1000);


  if(myFile)
  {

// Serial3.println("Strip0.txt");
    while(myFile.available())
    {
```

```
   Serial3.write(char(myFile.read()));
  }
  myFile.close();
}
```

```
//  myFile.close();
  }
  Serial3.flush();
}
```

AES – 128 Implementation on Teensy 3.1 Code

```
//      Instream...
int row,col,val, instream[4][4] =
{
  {0,0,0,0},
  {0,0,0,0},
  {0,0,0,0},
  {0,0,0,0}
};

//      S_BOX AND INVERSE S_BOX DEFINITION...

int sbox[16][16] =
{

{0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76},

{0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0},

{0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15},

{0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75},

{0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84},

{0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF},

{0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8},

{0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2},

{0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73},

{0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB},

{0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x9A,0x95,0xE4,0x79},

{0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08},

{0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A},

{0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E},

{0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF},

{0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16}
};

int invsbox[16][16] =
{

{0x52,0x09,0x6A,0xD5,0x30,0x36,0xA5,0x38,0xBF,0x40,0xA3,0x9E,0x81,0xF3,0xD7,0xFB},

{0x7C,0xE3,0x39,0x82,0x9B,0x2F,0xFF,0x87,0x34,0x8E,0x43,0x44,0xC4,0xDE,0xE9,0xCB},

{0x54,0x7B,0x94,0x32,0xA6,0xC2,0x23,0x3
```

```
D,0xEE,0x4C,0x95,0x0B,0x42,0xFA,0xC3,0
x4E},

{0x08,0x2E,0xA1,0x66,0x28,0xD9,0x24,0xB
2,0x76,0x5B,0xA2,0x49,0x6D,0x8B,0xD1,0x
25},

{0x72,0xF8,0xF6,0x64,0x86,0x68,0x98,0x16
,0xD4,0xA4,0x5C,0xCC,0x5D,0x65,0xB6,0x
92},

{0x6C,0x70,0x48,0x50,0xFD,0xED,0xB9,0x
DA,0x5E,0x15,0x46,0x57,0xA7,0x8D,0x9D,
0x84},

{0x90,0xD8,0xAB,0x00,0x8C,0xBC,0xD3,0x
0A,0xF7,0xE4,0x58,0x05,0xB8,0xB3,0x45,0
x06},

{0xD0,0x2C,0x1E,0x8F,0xCA,0x3F,0x0F,0x
02,0xC1,0xAF,0xBD,0x03,0x01,0x13,0x8A,0
x6B},

{0x3A,0x91,0x11,0x41,0x4F,0x67,0xDC,0xE
A,0x97,0xF2,0xCF,0xCE,0xF0,0xB4,0xE6,0
x73},

{0x96,0xAC,0x74,0x22,0xE7,0xAD,0x35,0x
85,0xE2,0xF9,0x37,0xE8,0x1C,0x75,0xDF,0
x6E},

{0x47,0xF1,0x1A,0x7A,0x1D,0x29,0xC5,0x
89,0x6F,0xB7,0x62,0x0E,0xAA,0x18,0xBE,0
x1B},

{0xFC,0x56,0x3E,0x4B,0xC6,0xD2,0x79,0x
20,0x9A,0xDB,0xC0,0xFE,0x78,0xCD,0x5A,
0xF4},

{0x1F,0xDD,0xA8,0x33,0x88,0x07,0xC7,0x
31,0xB1,0x12,0x10,0x59,0x27,0x80,0xEC,0x
5F},

{0x60,0x51,0x7F,0xA9,0x19,0xB5,0x4A,0x0
D,0x2D,0xE5,0x7A,0x9F,0x93,0xC9,0x9C,0
xEF},

{0xA0,0xE0,0x3B,0x4D,0xAE,0x2A,0xF5,0x
B0,0xC8,0xEB,0xBB,0x3C,0x83,0x53,0x99,
0x61},

{0x17,0x2B,0x04,0x7E,0xBA,0x77,0xD6,0x
26,0xE1,0x69,0x14,0x63,0x55,0x21,0x0C,0x
7D}
};

//      STATIC KEY DEFINITIONS...

//  Static Cipher Key:

int key1[4][4] =
{
  {0x41,0x43,0x45,0x48},
  {0x00,0x00,0x00,0x00},
  {0x42,0x25,0x76,0x49},
  {0x00,0x00,0x00,0x00}
};

//  Static Round Keys:

int key2[4][4] =
{
  {0x23,0x60,0x25,0x6D},
  {0x3B,0x3B,0x3B,0x3B},
  {0x21,0x04,0x72,0x3B},
  {0x52,0x52,0x52,0x52}
};

int key3[4][4] =
{
  {0xC3,0xA3,0x86,0xEB},
  {0xD9,0xE2,0xD9,0xE2},
  {0x21,0x25,0x57,0x6C},
  {0x6E,0x3C,0x6E,0x3C}
};

int key4[4][4] =
{
  {0x5F,0xFC,0x7A,0x91},
  {0x89,0x6B,0xB2,0x50},
  {0xCA,0xEF,0xB8,0xD4},
  {0x87,0xBB,0xD5,0xE9}
};

int key5[4][4] =
{
  {0x04,0xF8,0x82,0x13},
  {0xC1,0xAA,0x18,0x48},
  {0xD4,0x3B,0x83,0x57},
  {0x06,0xBD,0x68,0x81}
```

```
};

int key6[4][4] =
{
  {0x46,0xBE,0x3C,0x2F},
  {0x9A,0x30,0x28,0x60},
  {0xD8,0xE3,0x60,0x37},
  {0x7B,0xC6,0xAE,0x2F}
};

int key7[4][4] =
{
  {0xB6,0x08,0x34,0x1B},
  {0x00,0x30,0x18,0x78},
  {0xCD,0x2E,0x4E,0x79},
  {0x6E,0xA8,0x06,0x29}
};

int key8[4][4] =
{
  {0x4A,0x42,0x76,0x6D},
  {0xB6,0x86,0x9E,0xE6},
  {0x68,0x46,0x08,0x71},
  {0xC1,0x69,0x6F,0x46}
};

int key9[4][4] =
{
  {0x44,0x06,0x70,0x1D},
  {0x15,0x93,0x0D,0xEB},
  {0x32,0x74,0x7C,0x0D},
  {0xFD,0x94,0xFB,0xBD}
};

int key10[4][4] =
{
  {0xB6,0xB0,0xC0,0xDD},
  {0xC2,0x51,0x5C,0xB7},
  {0x48,0x3C,0x40,0x4D},
  {0x59,0xCD,0x36,0x8B}
};

int key11[4][4] =
{
  {0x29,0x99,0x59,0x84},
  {0x21,0x70,0x2C,0x9B},
  {0x75,0x49,0x09,0x44},
  {0x98,0x55,0x63,0xE8}
};

////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////

void setup()
{
  Serial.begin(9600);
  Serial3.begin(1382400);
  delay(3000);
}

////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
////////////////////////////////////////  MAIN
FUNCTION....
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////

void loop()
{
  for(col=0x0;col<4;col++)
  {
    for(row=0x0;row<4;row++)
    {
      if(Serial.available())
      {
        val = int(Serial.read());
        fromline(val,row,col);

        //Serial3.write(fromline(val,row,col));
      }

      /*if(Serial3.available())
      {
        Serial.write(Serial3.read());
      }*/
    }
  }

  delay(100);
}


////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
//////////////////////////////////////// AES
FUNCTIONS...
////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////
////////////////////////////////////////////////

///// Encryption ... /////

void fromline(int value, int i, int j)
{
  //char text;
  instream[i][j] = value;

  if(i==3 && j==3)
  {
    encrypt(instream);

    Serial.print("Ciphertext: ");

    for(j=0;j<4;j++)
    {
      for(i=0;i<4;i++)
      {
        //text = char(instream[i][j]);

        Serial.write(instream[i][j]);
        Serial3.write(instream[i][j]);
      }
    }
    Serial.println();

    decrypt(instream);

    Serial.print("Plaintext: ");

    for(j=0;j<4;j++)
    {
      for(i=0;i<4;i++)
      {
        //text = char(instream[i][j]);

        Serial.write(instream[i][j]);
        Serial3.write(instream[i][j]);
      }
    }

  }
}


void encrypt(int instream[][4])
{
  int i=0x0;

  addkey(instream,1);

  for(i=2;i<=10;i++)
  {
    subbytes(instream);
    shiftrows(instream);
    mixcol(instream);
    addkey(instream,i);
  }

  subbytes(instream);
  shiftrows(instream);
  addkey(instream,11);
}


void subbytes(int instream[][4])
{
  //  Test Matrix
  /*unsigned int instream[4][4] =
  {
    {0x19,0xa0,0x9a,0xe9},
    {0x3d,0xf4,0xc6,0xf8},
    {0xe3,0xe2,0x8d,0x48},
    {0xbe,0x2b,0x2a,0x08}
  };*/

  int i=0x0, j=0x0, MSN=0x0, LSN=0x0;

  for (j=0x00 ; j<0x04 ; j++)
    {
      for(i=0x00 ; i<0x04 ; i++)
      {
        MSN = ((0xf0 & instream[i][j]) >> 4);
        LSN = 0x0f & instream[i][j];

        instream[i][j] = sbox[MSN][LSN];
      }
    }

  /*for(i=0x0;i<4;i++)
  {
    for(j=0x0;j<4;j++)
    {       Serial.println(instream[i][j],HEX);}
  }*/

}
```

```
void shiftrows(int instream[][4])
{
  int i = 0x0, j=0x0, k=0x0, value=0x0;

  //  Test Matrix
   /*int instream[4][4] =
  {
    {0xd4,0xe0,0xb8,0x1e},
    {0x27,0xbf,0xb4,0x41},
    {0x11,0x98,0x5d,0x52},
    {0xae,0xf1,0xe5,0x30}
  };*/

  for(i=0x0;i<4;i++)
  {
    for(j=0x0;j<i;j++)
    {
      value = instream[i][0];

      for(k=0x0;k<3;k++)
      {
        instream[i][k] = instream[i][k+1];
      }

      instream[i][3] = value;
    }
  }

  /*Serial.println("shift...");
  for(i=0;i<4;i++)
  {
    for(j=0;j<4;j++)

{Serial.println(instream[i][j],HEX);delay(500
);}
    delay(500);
  }*/
}


void mixcol(int instream[][4])
{
  int i=0x0,j=0x0,k=0x0,sum=0x0;

  /*unsigned int instream[4][4] =
  {
    {0xd4,0xe0,0xb8,0x1e},
    {0xbf,0xb4,0x41,0x27},
    {0x5d,0x52,0x11,0x98},
    {0x30,0xae,0xf1,0xe5}
  };*/

    int outstream[4][4] =
  {
    {0x0,0x0,0x0,0x0},
    {0x0,0x0,0x0,0x0},
    {0x0,0x0,0x0,0x0},
    {0x0,0x0,0x0,0x0}
  };                          // All values initialized
to zero value.

  int Ammar[16] = {0},ct = 0x0;

  int mixer[4][4] =
  {
    {0x02,0x03,0x01,0x01},
    {0x01,0x02,0x03,0x01},
    {0x01,0x01,0x02,0x03},
    {0x03,0x01,0x01,0x02}
  };


  for(j=0x0;j<0x4;j++)
  {
    for(i=0x0;i<0x4;i++)
    {

      for(k=0x0;k<0x4;k++)
      {
        if(mixer[i][k] == 0x02)
        {outstream[k][j] =
mult2(instream[k][j]);}
        else if(mixer[i][k] == 0x03)
        {outstream[k][j] =
mult3(instream[k][j]);}
        else
        {outstream[k][j] = instream[k][j];}

        sum = sum ^ outstream[k][j];
        //Serial.println(sum,BIN);

        outstream[i][j] = sum;

        //delay(200);
      }
        Ammar[ct] = outstream[i][j];
        ct++;

        /*Serial.print("i=");
```

```
      Serial.print(i,DEC);
      Serial.print("j=");
      Serial.println(j,DEC);
      Serial.println(outstream[i][j],HEX);
      Serial.println(outstream[0][0],HEX);*/

    sum = 0x0;

    outstream[i][j] = 0;

    //Serial.println();
   }
  }

ct = 0;

for(i=0x0;i<0x4;i++)
{
for(j=0x0;j<0x4;j++)
{outstream[j][i] = Ammar[ct]; ct++;} //
Serial.println(outstream[j][i],HEX);}
}

}



void addkey(int instream[][4], int rnd_no)
{
  int i=0x0,j=0x0;

  switch (rnd_no)
  {
    case 1:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key1[i][j];
         }
       }
    break;
    }

    case 2:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key2[i][j];
         }
       }
    break;
    }

    case 3:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key3[i][j];
         }
       }
    break;
    }

    case 4:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key4[i][j];
         }
       }
    break;
    }

    case 5:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key5[i][j];
         }
       }
    break;
    }

    case 6:
    {
       for(i=0x0;i<4;i++)
```

```
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key6[i][j];
       }
      }
   break;
   }

   case 7:
   {
      for(i=0x0;i<4;i++)
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key7[i][j];
       }
      }
   break;
   }

   case 8:
   {
      for(i=0x0;i<4;i++)
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key8[i][j];
       }
      }
   break;
   }

   case 9:
   {
      for(i=0x0;i<4;i++)
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key9[i][j];
       }
      }
   break;
   }

   case 10:
   {
      for(i=0x0;i<4;i++)
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key10[i][j];
       }
      }
   break;
   }

   case 11:
   {
      for(i=0x0;i<4;i++)
      {
       for(j=0x0;j<4;j++)
       {
          instream[i][j] = instream[i][j] ^
key11[i][j];
       }
      }
   break;
   }

  default: {Serial.println("Add Key - Logical
Error...");  }
  }
}


///// Auxiliary... //////


int mult2(int x)
{
  if((x&B10000000) == B10000000)
    {

      x = x<<1;
      x = x & 0x0ff;
      x = x | 0x1b;

    }

  else
  {
     x = x<<1;
  }
```

```
  return x;
}

int mult3(int x)
{
  int sum = x ^ mult2(x);
  return sum;
}


int multe(int x)
{
  int sum = mult2(x);

  sum = sum ^ mult2(mult2(x));

  sum = sum ^ mult2(mult2(mult2(x)));

  return sum;
}


int multb(int x)
{
  int sum = x;

  sum = sum ^ mult2(x);

  sum = sum ^ mult2(mult2(mult2(x)));

  return sum;
}


int multd(int x)
{
  int sum = x;

  sum = sum ^ mult2(mult2(x));

  sum = sum ^ mult2(mult2(mult2(x)));

  return sum;
}


int mult9(int x)
{
  int sum = x;

  sum = sum ^ mult2(mult2(mult2(x)));

  return sum;
}


///// Decryption... /////

void decrypt(int instream[][4])
{
  int i=0x0;

  invaddkey(instream,1);

  for(i=2;i<=10;i++)
  {
    invshiftrows(instream);
    invsubbytes(instream);
    invaddkey(instream,i);
    invmixcol(instream);
  }

  invshiftrows(instream);
  invsubbytes(instream);
  invaddkey(instream,11);
}


void invsubbytes(int instream[][4])
{
  //  Test Matrix
  /*unsigned int instream[4][4] =
  {
    {0x19,0xa0,0x9a,0xe9},
    {0x3d,0xf4,0xc6,0xf8},
    {0xe3,0xe2,0x8d,0x48},
    {0xbe,0x2b,0x2a,0x08}
  };*/

  int i=0x0, j=0x0, MSN=0x0, LSN=0x0;

  for (j=0x00 ; j<0x04 ; j++)
    {
      for(i=0x00 ; i<0x04 ; i++)
      {
        MSN = ((0xf0 & instream[i][j]) >> 4);
        LSN = 0x0f & instream[i][j];

        instream[i][j] = invsbox[MSN][LSN];
      }
```

```
   }

 /*for(i=0x0;i<4;i++)
 {
  for(j=0x0;j<4;j++)
  {       Serial.println(instream[i][j],HEX);}
 }*/

}


void invshiftrows(int instream[][4])
{
 int i = 0x0, j=0x0, k=0x0, value=0x0;

 //  Test Matrix
 /* unsigned int plain[4][4] =
 {
  {0xd4,0xe0,0xb8,0x1e},
  {0x27,0xbf,0xb4,0x41},
  {0x11,0x98,0x5d,0x52},
  {0xae,0xf1,0xe5,0x30}
 };*/

 for(i=0x0;i<4;i++)
 {
  for(j=0x0;j<i;j++)
  {
   value = instream[i][3];

   for(k=3;k>0;k--)
   {
    instream[i][k] = instream[i][k-1];
   }

   instream[i][0] = value;
  }
 }

  /* Serial.println("inv_shift...");
 for(i=0;i<4;i++)
 {
  for(j=0;j<4;j++)

{Serial.println(instream[i][j],HEX);delay(500
);}
   delay(500);
 }*/
}


void invmixcol(int instream[][4])
{
 int i=0x0,j=0x0,k=0x0,sum=0x0;

 /*unsigned int instream[4][4] =
 {
  {0xd4,0xe0,0xb8,0x1e},
  {0xbf,0xb4,0x41,0x27},
  {0x5d,0x52,0x11,0x98},
  {0x30,0xae,0xf1,0xe5}
 };*/

  int outstream[4][4] =
 {
  {0x0,0x0,0x0,0x0},
  {0x0,0x0,0x0,0x0},
  {0x0,0x0,0x0,0x0},
  {0x0,0x0,0x0,0x0}
 };                          // All values initialized
to zero value.

 int Ammar[16] = {0},ct = 0x0;

 int mixer[4][4] =
 {
  {0x0e,0x0b,0x0d,0x09},
  {0x09,0x0e,0x0b,0x0d},
  {0x0d,0x09,0x0e,0x0b},
  {0x0b,0x0d,0x09,0x0e}
 };


 for(j=0x0;j<0x4;j++)
 {
  for(i=0x0;i<0x4;i++)
  {

    for(k=0x0;k<0x4;k++)
    {
     if(mixer[i][k] == 0x0e)
     {outstream[k][j] =
multe(instream[k][j]);}
      else if(mixer[i][k] == 0x0b)
      {outstream[k][j] =
multb(instream[k][j]);}
      else if(mixer[i][k] == 0x0d)
      {outstream[k][j] =
multd(instream[k][j]);}
      else if(mixer[i][k] == 0x09)
```

```
      {outstream[k][j] =
mult9(instream[k][j]);}
      else
      {outstream[k][j] = instream[k][j];}

      sum = sum ^ outstream[k][j];
      //Serial.println(sum,BIN);

      outstream[i][j] = sum;

      //delay(200);
     }
      Ammar[ct] = outstream[i][j];
      ct++;

      /*Serial.print("i=");
      Serial.print(i,DEC);
      Serial.print("j=");
      Serial.println(j,DEC);
      Serial.println(outstream[i][j],HEX);
      Serial.println(outstream[0][0],HEX);*/

     sum = 0x0;

     outstream[i][j] = 0;

     //Serial.println();
    }
  }

ct = 0;

for(i=0x0;i<0x4;i++)
{
for(j=0x0;j<0x4;j++)
{outstream[j][i] = Ammar[ct]; ct++;} //
Serial.println(outstream[j][i],HEX);}
}

}


void invaddkey(int instream[][4], int rnd_no)
{
  int i=0x0,j=0x0;

  switch (rnd_no)
  {
    case 1:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key11[i][j];
         }
       }
    break;
    }

    case 2:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key10[i][j];
         }
       }
    break;
    }

    case 3:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key9[i][j];
         }
       }
    break;
    }

    case 4:
    {
       for(i=0x0;i<4;i++)
       {
         for(j=0x0;j<4;j++)
         {
            instream[i][j] = instream[i][j] ^
key8[i][j];
         }
       }
    break;
    }
```

```
   case 5:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key7[i][j];
        }
      }
   break;
   }

   case 6:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key6[i][j];
        }
      }
   break;
   }

   case 7:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key5[i][j];
        }
      }
   break;
   }

   case 8:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key4[i][j];
        }
      }
   break;
   }

   case 9:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key3[i][j];
        }
      }
   break;
   }

   case 10:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key2[i][j];
        }
      }
   break;
   }

   case 11:
   {
      for(i=0x0;i<4;i++)
      {
        for(j=0x0;j<4;j++)
        {
           instream[i][j] = instream[i][j] ^
key1[i][j];
        }
      }
   break;
   }

  default: {Serial.println("Inv Add Key -
Logical Error...");  }
  }
}
```

BIBLOGRAPHY

[1] Sandisk, "News Release," 9 5 2014. [Online]. Available: http://www.sandisk.com/about-sandisk/press-room/press-releases/2008/2008-04-09-sandisk-survey-shows-organizations-at-risk-from-unsecured-usb-flash-drivesusage-is-more-than-double-corporate-it-expectations/.

[2] S. Keller, "Wireless USB," Cornell University, 2004.

[3] Universal Serial Bus, "USB 3.1 Specification," [Online]. Available: http://www.usb.org/developers/docs/usb_31_031114.zip. [Accessed 11 05 2014].

[4] Universal Serial Bus, "USB 2.0 Documents Index," [Online]. Available: http://www.usb.org/developers/docs/usb20_docs/usb_20_042814.zip. [Accessed 11 05 2014].

[5] Bluetooth, "Specification Adopted Documents," [Online]. Available: https://www.bluetooth.org/en-us/specification/adopted-specifications. [Accessed 11 5 2014].

[6] SanDisk, "SanDisk Connect Wireless Flash Drive," SanDisk, [Online]. Available: http://www.sandisk.com/products/wireless/flash-drive/. [Accessed 11 5 2014].

[7] Kingston, "Wireless Drives," Kingston, [Online]. Available: http://www.kingston.com/en/wireless/wireless_drives. [Accessed 11 5 2014].

[8] Toshiba, "FlashAir™ - SD Card with Embedded Wireless LAN," Toshiba, [Online]. Available: http://www.toshiba-components.com/FlashAir/. [Accessed 11 5 2014].

[9] PJRC, "PJRC Electronics Projects Components Available Worldwide," [Online]. Available: https://www.pjrc.com/teensy/teensy31.html. [Accessed 2014].