SECURE SD CARD READER

By

CAPT MUHAMMAD UMAIR SIDDIQUI

CAPT MUHAMMAD UMAR KHAN

CAPT RANA EHTESHAM-UL-HAQ

Submitted to Department of Electrical Engineering, Military College of Signals National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of a B.E Degree in Telecom Engineering
June 2014

ABSTRACT
SECURE SD CARD READER

With countless portable devices easily available such as SD-Cards, CDs, Flash drives and MP3 playersas well as Internet access on nearly every workstation, businesses run an increased risk of data theft by employees and others with access to confidential data. Data theft continues to rise and create problems with companies every day. Data theft is a term used to describe when information is unlawfully copied or taken from a business or other individual. SD-Card widespread use sometimes makes them relatively safer than they are.

In fact, even popular and advantageous innovative devices making use of SD-Card may have significant risks. Not surprisingly, if the portable device or media contains tax returns or other sensitive information, you could be ethically, legally and financially responsible for a security breach and its consequences. The danger is too great to ignore, but these convenient devices have become a tactical necessity for everyone on the go.

Secure SD-Card reader mitigates the risks of portable data storage devices so you can use them without hesitation. The SD-Card reader uses advanced encryption standard for encryption and decryption of data. The project will be completed in two phases in first phase read/write from SD-card will take place using microcontroller and in second phase we will implement Advanced Encryption Standard for the encryption of data.

CERTIFICATE OF CORRECTNESS AND APPROVAL

It is certified that the work contained in this thesis titled “Secure SD Card Reader”, carried out by Muhammad Umair Siddiqui, Muhammad Umar Khan and RanaEhtesham-Ul-Haq under the supervision of Lt ColIshtiaqKiyani in partial fulfillment of the Bachelors of Telecommunication Engineering, is correct and approved.

Approved By

_____

Asst. Prof. IshtiaqKiyani

Project Supervisor

Military College of Signals, NUST

DEDICATION

Almighty Allah,

Our parents for their prayers

And Faculty for their help

ACKNOWLEDGEMENTS

Nothing happens without the will of Allah Almighty. Special thanks to Allah Almighty for giving us knowledge and strength to accomplish this task successfully.

The team likes to thank our project supervisor, Lt ColIshtiaqKiyani, without his support and encouragement; it would not have been possible to complete this project.

Special thanks to our parents for their unrelenting support and belief in us and for their prayers.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

List of Abbreviations

SD	Secure Digital
SDHC	Secure Digital High Capacity
CD	Compact Disk
MP3	MPEG-1 Part 3 (Moving Picture Expert Group)
LED	Light Emitting Diode
USB	Universal Serial Bus
HDD	Hard Disk Drive
PWM	Pulse Width Modulation
UART	Universal Asynchronous Receiver/Transmitter
ICSP	In Circuit Serial Programming
RISC	Reduced Instruction Set Computing
FAT	File Allocation Table
SRAM	Static Random Access Memory
MPU	Memory Protection Unit
PDCA	Peripheral Direct Memory Access Controller
DMACA	Direct Memory Access controller
SDIO	Secure Digital Input Output
MMC	MultiMediaCard
GPIO	General purpose Input Output
SPI	Serial to parallel Interface
NIST	National Institute of Standards and Technology
AES	Advanced Encryption Standard

1. INTRODUCTION

1.1 OVERVIEW

Data theft continues to rise and create problems with companies every day.

It is important to understand the possibilities and consequences of data theft. It is also important to understand how to prevent data theft. Data theft is a growing problem primarily perpetrated by office workers with access to technology such as desktop computers and hand-held devices capable of storing digital information such as USB flash drives, SD Cards, iPods and even gadgetry using memory cards. Since employees often spend a considerable amount of time developing contacts and confidential and copyrighted information for the company they work for, they often feel they have some right to the information and are inclined to copy and/or delete part of it when they leave the company, or misuse it while they are still in employment. While most organizations have implemented firewalls and intrusion-detection systems very few take into account the threat from the average employee that copies proprietary data for personal gain or use by another company. A common scenario is where a sales person makes a copy of the contact database for use in their next job. Typically this is a clear violation of their terms of employment. Unfortunately, data theft is rising every day. Steps must be taken to assure proper protection. This motivation to protect the data lead us to make a device that will encrypt the data before storing it in a portable device and decrypt the data before retrieving it.

1.2 PROJECT DESCRIPTION

The purpose of this project is to secure data from unauthorized access. Data Security means protecting a database from destructive forces and the unwanted actions of unauthorized users. One common approach is to encrypt the data for storage, and then transfer that storage media. In this way only the authorize users will have access to the data.

It's worth highlighting an often-underappreciated advantage of encrypting data on portable storage devices. Specifically, properly encrypted data offers a safety net against potentially embarrassing or damaging data surfacing from storage devices that were discarded or sold off. Many businesses don't realize how easily deleted files can be retrieved with off-the-shelf recovery software from mechanical storage devices such as SD-cards, hard disk drives (HDD) or USB drives. Reconstituting previously encrypted data, on the other hand, is far more difficult, as it that requires the original credentials or even a copy of the decryption key. An encrypted storage device with a decryption key that's been erased, or one with a good authentication passphrase, offers a good safeguard against malicious data recovery. A thoroughly wiped or physically destroyed storage device remains the most secure defense against data leakage, though.

Data integrity refers to maintaining and assuring the accuracy and consistency of data over its entire life-cycle. The purpose of this project is also to ensure integrity of data stored in SD-card. Since only the authorize users have access to the contents of the memory card, no one can change the data without authentication. It also restricts auto run viruses to move into SD-card and damages everything on the card and stop further replication of auto run virus on other computers.

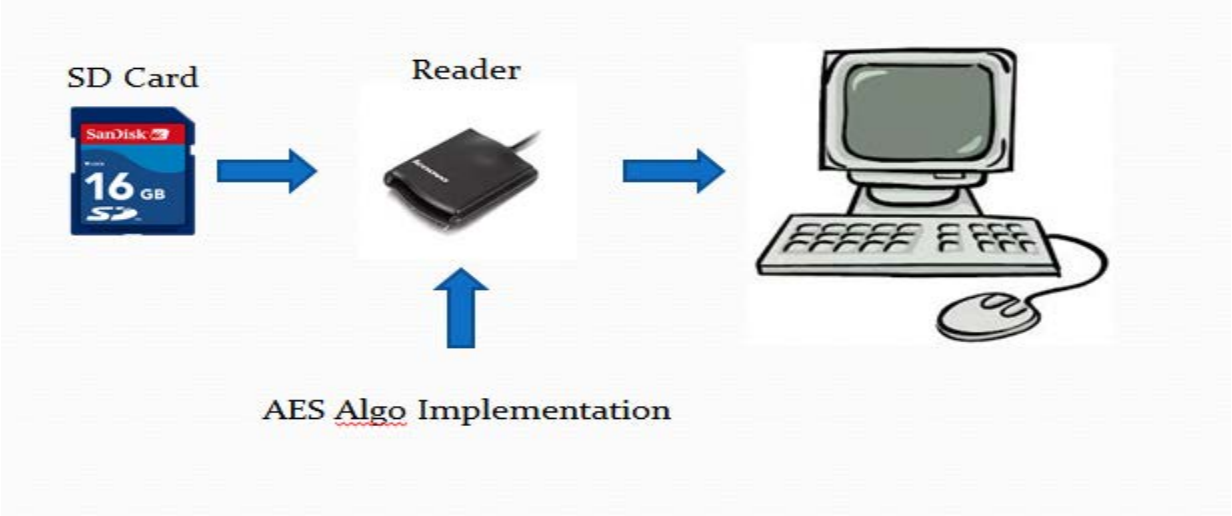


Figure 1-1 Block Diagram

Our device will be capable of reading and writing from SD Card in a secure manner. In order to achieve that security and protection we have implemented AES encryption on it. All the data that passes from computer to the SD card will be encrypted using AES encryption and on the other hand all the data from SD card to computer will be decrypted to use it.

1.3 SCOPE

The scope of our project is to make a SD-Card reader which will encrypt the data before storing it in a SD Card and decrypt the data before using it again with the help of Advanced Encryption Standard (AES).

2. LITERATURE REVIEW

In the first step read and write from SD Card will have to be done for this purpose some open source hardware computing platforms were used some of them are:-

2.1ARDUINO MEGA2560

The Arduino Mega is a microcontroller board based on the ATmega1280. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.[1]

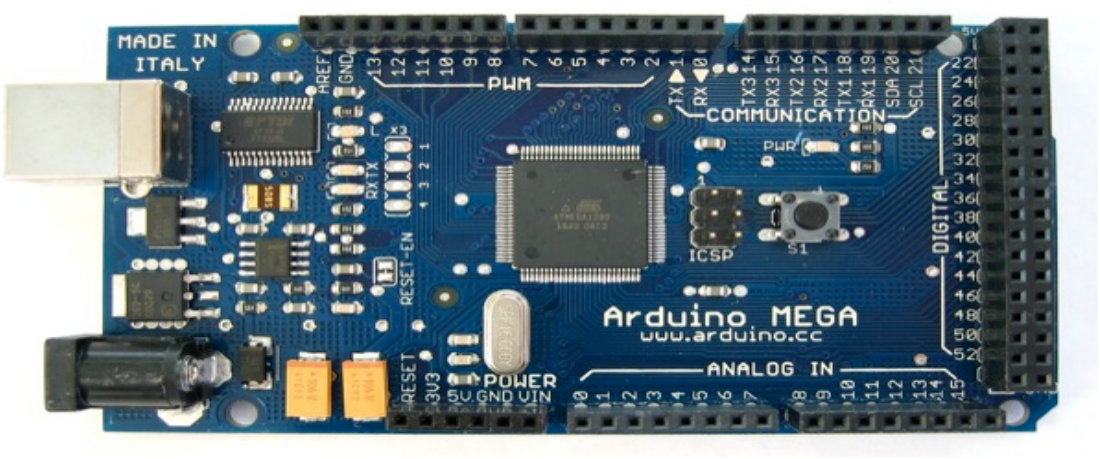


Figure 2-1 Arduino mega2560

Summary

Microcontroller	ATmega1280
Operating Voltage	5V
Input Voltage	7-12V

(recommended)	
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
Flash Memory	128 KB of which 4 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

2.2AT32UC3A3218S

The AT32UC3A3/A4 is a complete System-On-Chip microcontroller based on the AVR32 UC RISC processor running at frequencies up to 84MHz. AVR32 UC is a high-performance 32-bit RISC microprocessor core, designed for cost-sensitive embedded applications, with particular emphasis on low power consumption, high code density and high performance. The processor implements a Memory Protection Unit (MPU) and a fast and flexible interrupt controller for supporting modern operating systems and real-time operating systems. Higher computation capabilities are achievable using a rich set of DSP instructions. The AT32UC3A3218 incorporates on-chip Flash and SRAM memories for secure and fast access. 64 KBytes of SRAM are directly coupled to the AVR32 UC for performances optimization. The Peripheral Direct Memory Access Controller (PDCA) enables data

transfers between peripherals and memories without processor involvement. The PDCA drastically reduces processing overhead when transferring continuous and large data streams. The peripheral set includes a High Speed MCI for SDIO/SD/MMC and a hardware encryption module based on AES algorithm. The Direct Memory Access controller (DMACA) allows high bandwidth data flows between high speed peripherals (USB, External Memories, MMC, SDIO) and through high speed internal features (AES). The Embedded Host interface allows device like a USB Flash disk or a USB printer to be directly connected to the processor. This peripheral has its own dedicated DMA and is perfect for Mass Storage application.[2]

Features:-

- High Performance, Low Power 32-bit Microcontroller
- Internal High-Speed Flash
- External Storage device support
- Buffer Encryption/Decryption Capabilities
- Support for SPI
- Processing Speed of 84 MHz
- Single 3.3V Power Supply

2.3 ATMEGA1281

The ATmega1281 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega1281 achieves throughputs approaching 1 MIPS per MHz allowing the system designed to optimize power consumption versus processing speed. Its main features are:-[3]

Features:-

- High Performance
- Low Power 8-Bit Microcontroller
- 16 MIPS Throughput at 16MHz
- 128KBytes of In-System Self-Programmable Flash
- 8Kbytes Internal SRAM
- Two Programmable Serial USART
- Master/Slave SPI Serial Interface
- 54 Programmable I/O Lines

In our project we will encrypt data using AES Algorithm. The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is based on the Rijndael cipher developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST during the AES selection process. Rijndael is a family of ciphers with different key and block sizes. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.

2.4 HIGH-LEVEL DESCRIPTION OF AES

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.[4]

2.4.1 Key Expansion

Round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more. The AES algorithm takes the Cipher Key and performs a Key Expansion routine to generate a key schedule. The algorithm requires an initial set of 4 words, and each round requires 4 words of key data. The resulting key schedule consists of a linear array of 4-byte words. First we applied RotWord() function to the last word. The function RotWord() takes a word [a0,a1,a2,a3] as input, performs a cyclic permutation, and returns the word [a1,a2,a3,a0]. Then we carry out SubWord() function. The SubWord() is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word. This output word is now XORed with the first word and first word of Rcon[] array. The Rcon[i] is a round constant word array. The first word of the expanded key are filled with the Cipher Key. Every following word is equal to the XOR of the previous word, and the word 4 positions earlier. In this way we have formed key matrix for the one round and for subsequent rounds last word is taken and repeat the process.[5]

2.4.2 Encryption

2.4.2.1 Initial Round

In initial round AddRoundKey function is performed where each byte of the state is combined with a block of the round key using bitwise XOR. As shown in the figure 2-2.[6]

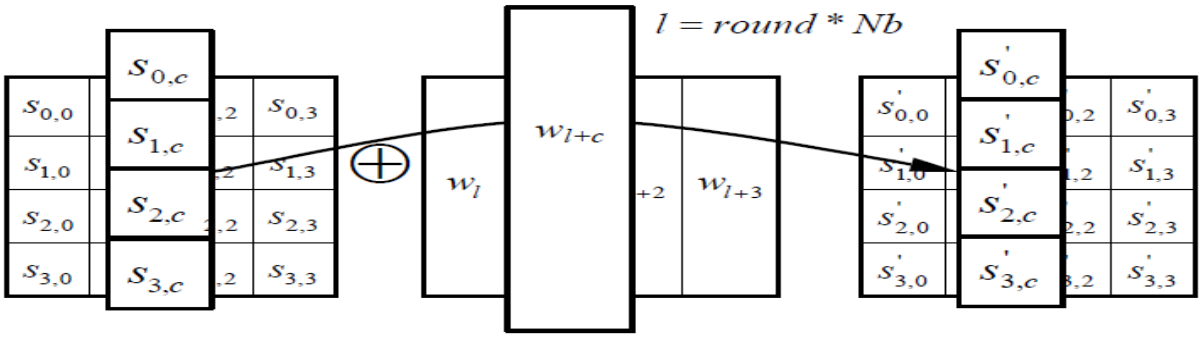


Figure 2-2 Add Round Key Function

2.4.2.2 Rounds

Four functions are performed in each round of AES, for AES of 128 bit the block size of data is 128 bit and there are 10 rounds in it.[7]

2.4.2.3 SubBytes

A non-linear substitution step where each byte is replaced with another according to a lookup table 2-1.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2-1Sub Bytes S-Box

2.4.2.4 ShiftRows

A transposition step where the last three rows of the state are shifted cyclically a certain number of steps as shown in figure 2-3.

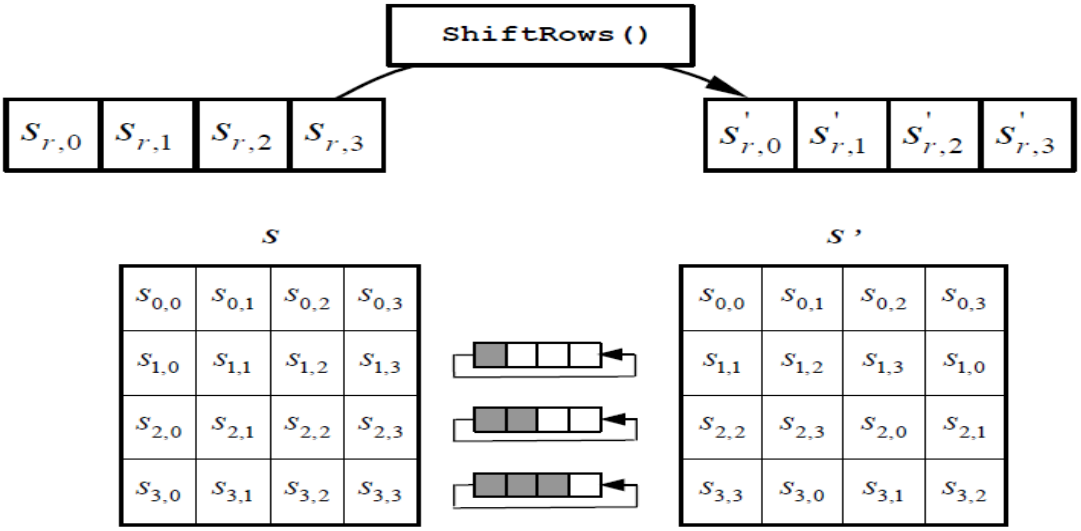


Figure 2-3 Shift Rows

2.4.2.5Mix Columns

A mixing operation which operates on the columns of the state, combining the four bytes in each column as shown in figure 2-4.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 2-4 Mix Coloumn

2.4.2.6 AddRoundKey

Same as performed in initial round.

2.4.2.7 Final Round

In final round no mix column function is performed. Only SubBytes, ShiftRows and AddRoundKey functions are performed.

2.4.3 Decryption

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher are Inverse ShiftRows, Inverse SubBytes,Inverse MixColumns, and AddRound Key.[7]

2.4.3.1 Inverse Shift Rows

Inverse ShiftRows is the inverse of the ShiftRows transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes. The first row, $r = 0$, is not shifted. The bottom three rows are cyclically shifted by the row numberas shown in figure 2-5.

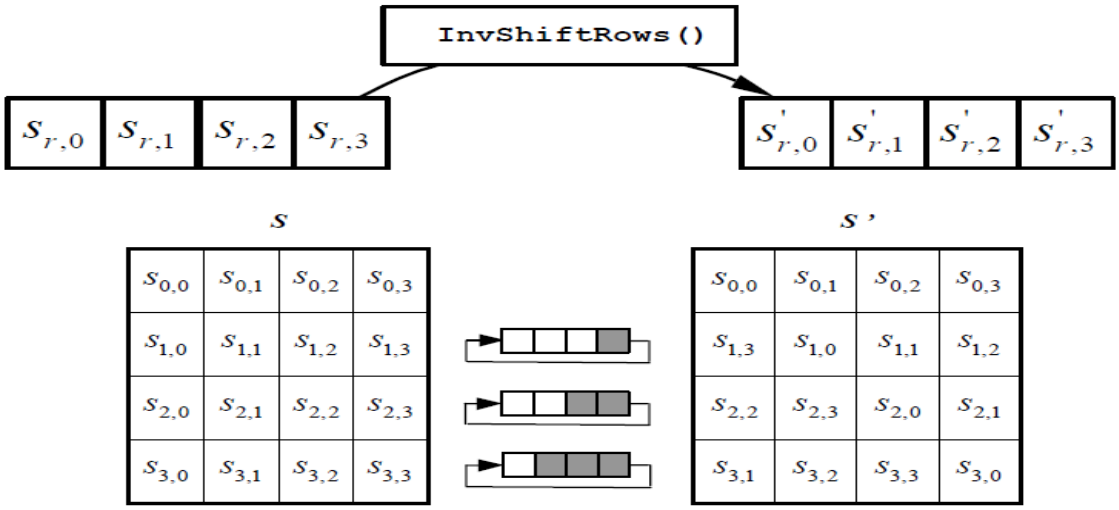


Figure 2-5 Inverse ShiftRows

2.4.3.2 Inverse SubBytes

Inverse SubBytesis the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the Statetable 2-3.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Table 2-2 Inverse Sub Bytes S-Box

2.4.3.3 Inverse Mix Columns

Inverse Mix Columnsis the inverse of the MixColumns transformation. Inverse MixColumns operates on the State column-by-columnas shown in figure 2-6.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 2-6 Inverse Column Matrix

2.4.3.4 AddRoundKey

Add round key function which was described earlier, is its own inverse, since it only involves an application of the XOR operation.

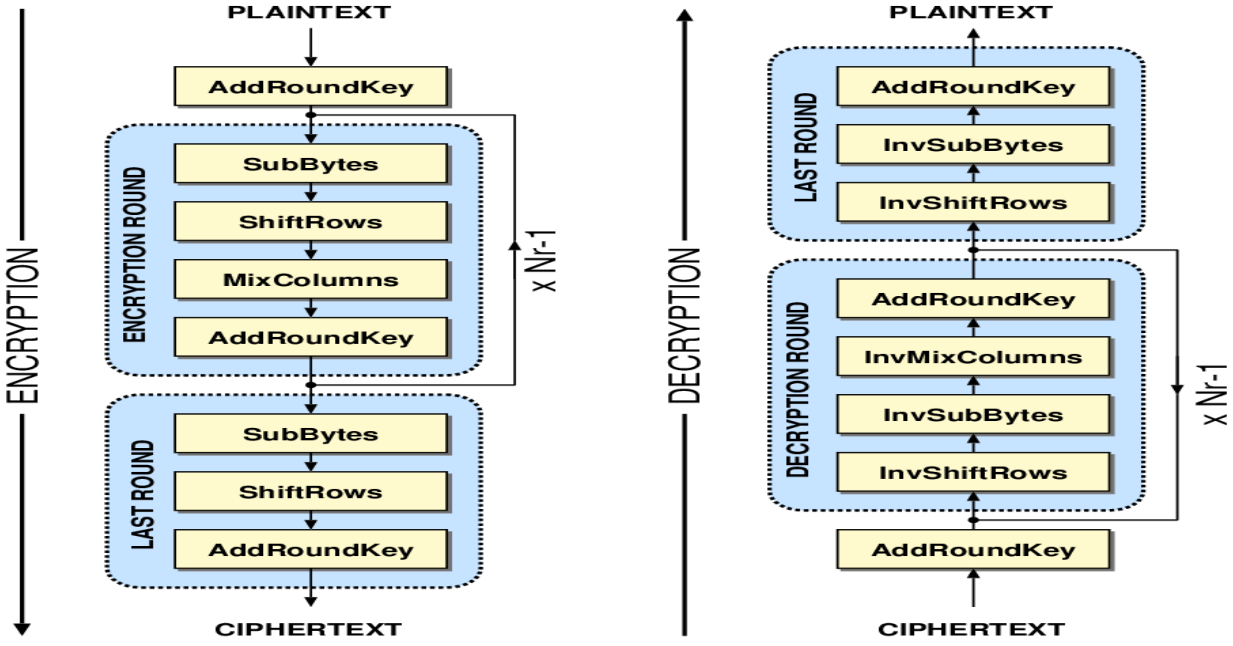


Figure 2-7 AES Algorithm

3. DESIGN

3.1 INTRODUCTION

Our project is divided into two parts first we have to read/write from SD card and then we have to implement AES on it. For this purpose first we have used smart board Arduino mega and then designed our own circuit using ATMega1281 microcontroller.

3.2 ARDUINO MEGA

We have used Arduino mega smart board to access SD Card and read write files from it. For this purpose we have used Arduino Ethernet shield to access SD Cardshown in figure 3-1. [8]

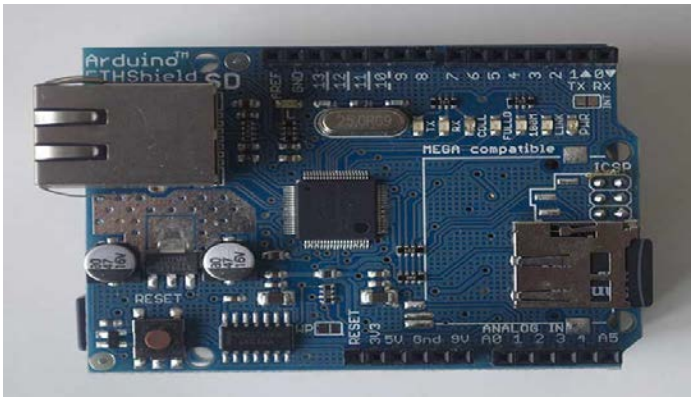


Figure 3-1 Arduino Ethernet Shield

The code for read/write from SD card is attached as per appendix ‘C’.

3.3 CIRCUIT DESIGN AND IMPLEMENTATION

As Arduino mega is a smart board and it includes many other accessories which are not required. In order to meet our desired design we used ATMega1281 which is same micro controller used in Arduino board and then made our own circuit. We have made all the simulations and circuit design on proteus. Circuit as shown in figure 3-2

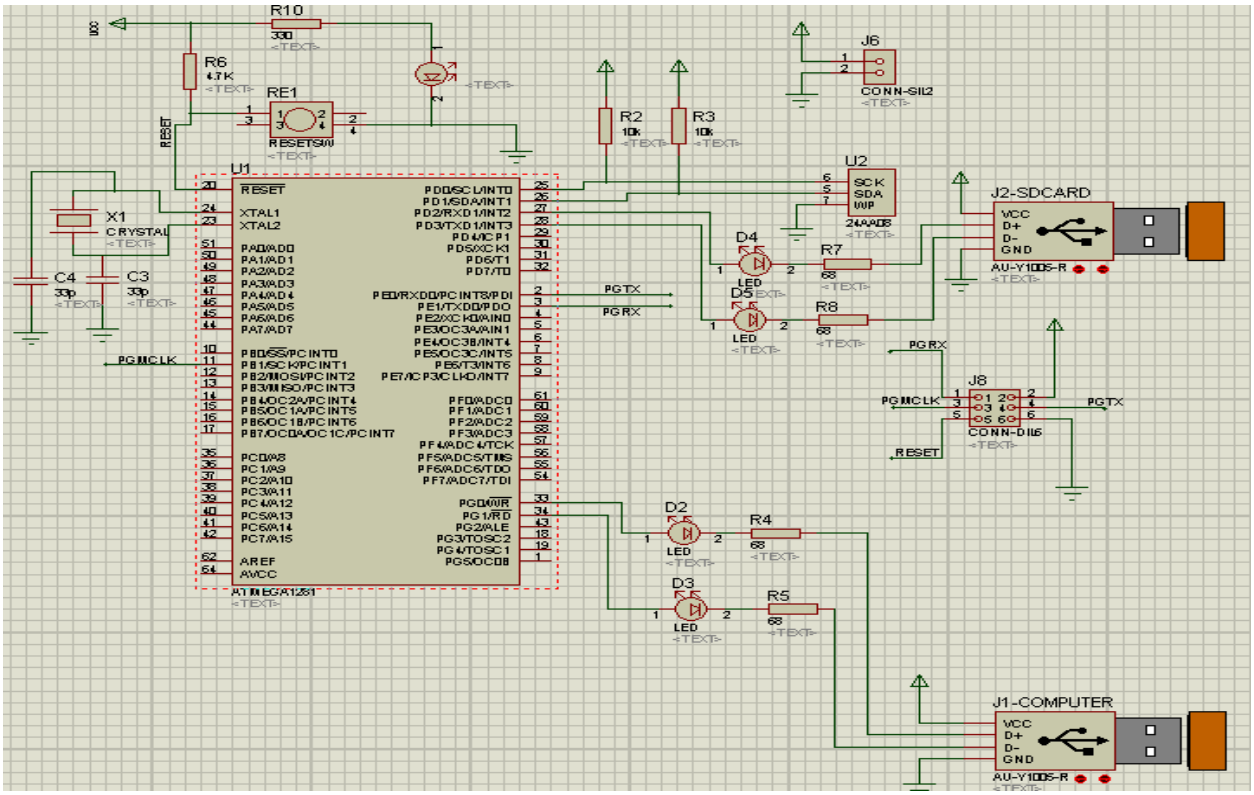


Figure 3-2 Circuit Diagram of Device

3.3.1 Accessing SD Card

ATMega1281 is a microcontroller which has 7 I/O ports namely from port A to port G. We used port D to transmit data to SD card for this we use pin 28 of microcontroller to transmit data and pin 27 to receive data. [9]

3.3.2 Adding External Memory

We have added 8 Kbit Electrically Erasable PROM to increase our memory requirement. We have connected serial data pin of chip 24AA08 on pin 26 and serial clock of this chip on pin 25.

3.3.3 Reading / Writing From Computer

We have used port G of microcontroller to read and write data from computer. For this we used pin 33 to write data and pin 34 to read data through USB.

3.3.4 Reset

We have added a button to reset the microcontroller and for this we uses reset pin of controller that is pin 20.

3.3.5 Oscillator

We have attached 16 MHz crystal oscillator on pin 23 and pin 24 of the controller.

3.3.6 Implementation

We have used Atmel Studio 6.1 to code our microcontroller. The code for implementation of AES is attached as per appendix ‘D’ and code for read/write from SD Card is attached as per Annex ‘E’.

3.3.7 GUI

We have designed a GUI of working of our project in c# visual basic to show the functionality of our project on computer.

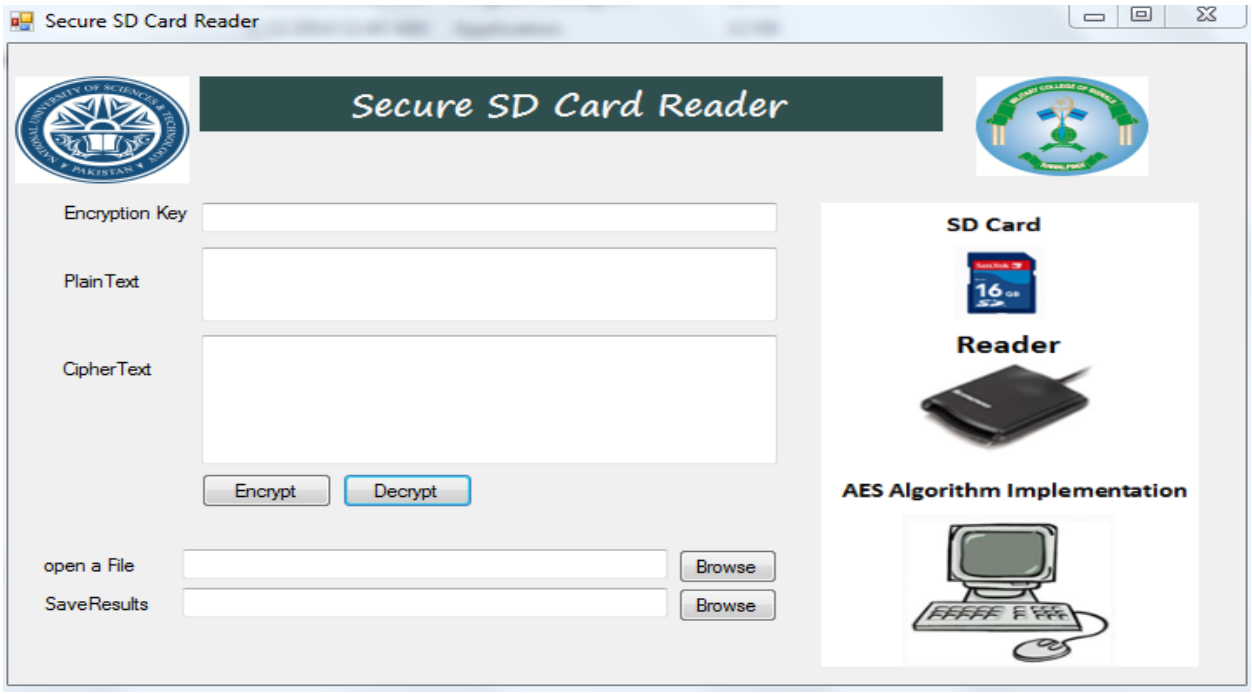


Figure 3-3 Application

4. PROJECT ANALYSIS AND EVALUATION

4.1 RESULTS AND ANALYSIS

From the results and output obtained from the (Proteus) simulation implementation of AES is confirmed. Hardware design has also been completed. Our aim was to make a device that can secure data from unauthorized access while moving data from one place to another. We have successfully implemented encryption of data before storing it on a SD card and decrypting it before accessing it again.

Overall project was in working condition but due to small memory available and slow processing speed of microcontroller we cannot able to encrypt large file with our design, however using a 32 bit microcontroller with more space and high processing speed we can also encrypt and decrypt large files. In our project we have used a fix key for encryption and decryption this means only one reader can

access the data which has encrypted it. We have implemented AES128 bit for the encryption of data.

5. RECOMMENDATIONS/ FUTURE WORK

Following are recommended for improvement of the project in future:

- Amendments may be made in AES algorithm for example changing the values of S-Box, changing the sequence of round functions will make AES cipher difficult to break.
- Instead of using a fix key, adding a keypad on device will facilitate everyone to select its own key to encrypt the data and further enhance the security.
- Micro controller of 16 bit or more should be used for fast processing and encrypting large capacity file. This will also solve our problem of adding an external memory.

6. CONCLUSION

The purpose of this project is to secure data from unauthorized access. All the data that passes from computer to the SD card will be encrypted using AES encryption and on the other hand all the data from SD card to computer will be decrypted to use it.

6.1 OBJECTIVE

We have successfully achieved our aim of securing data using AES algorithm and making of hardware device. However our device it is not able to secure large files due to low processing speed and less memory of microcontroller.

6.2 APPLICATIONS

Data safety and integrity is one of the main issues of organizations and this device can be used to avoid unauthorized access and alteration in the data. Encrypted Data is stored in SD card so data recovery tools can only recovered the encrypted data which is of no use without authentic pass key.

6.3 Limitations

Microcontroller of 32 bit and its accessories are not easily available, which forced us to select 8-bit microcontroller which has slow processing speed and low memory. Programmer for microcontroller is also not available.

7. BIBLIOGRAPHY

1. *Arduino - ArduinoBoardMega2560*. (n.d.). Retrieved from http://arduino.cc/en/Main/arduinoBoardMega2560

2. *AT32UC3A3128S*. (n.d.). Retrieved from http://www.atmel.com/devices/at32uc3a3128s.aspx

3. *ATmega1281*. (n.d.). Retrieved from http://www.atmel.com/devices/atmega1281.aspx

4. *Advanced Encryption Standard - Wikipedia, the free encyclopedia*. (n.d.). Retrieved March 25, 2014, from http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

5. *NIST Computer Security Division - Cryptographic Toolkit*. (n.d.). Retrieved from http://csrc.nist.gov/groups/ST/toolkit/index.html

6. *block cipher - AES AddRoundKey - Cryptography Stack Exchange*. (n.d.). Retrieved from http://crypto.stackexchange.com/questions/8043/aes-addroundkey

7. *NIST.gov - Computer Security Division - Computer Security Resource Center*. (n.d.). Retrieved from http://csrc.nist.gov/groups/ST/toolkit/key_management.html

8. *How to Interface SD Card with Arduino: Arduino SD Card Project with Circuit Diagram*. (n.d.). Retrieved from http://www.engineersgarage.com/embedded/arduino/how-to-interface-sd-card-with-arduino-project-circuit

9. *Simple SD Card Read/Write Setup*. (n.d.). Retrieved from http://wiki.dxarts.washington.edu/groups/general/wiki/1c44f/Simple_SD_Card_ReadWrite_Setup.html

10. *Secure Digital - Wikipedia, the free encyclopedia*. (n.d.). Retrieved March 25, 2014, from http://en.wikipedia.org/wiki/Secure_Digital

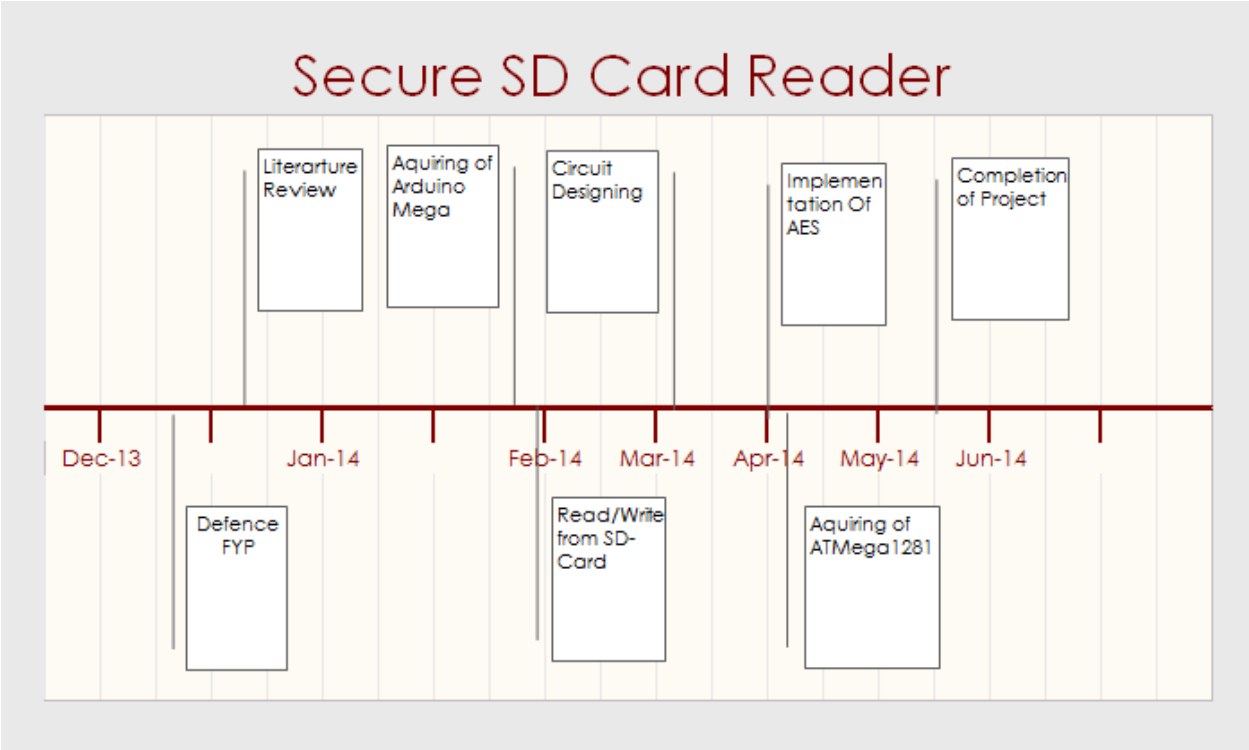
Appendix 'A'

COST ON PROJECT

SERIAL	WORK DONE	COST
1	Purchase & Shipment of Arduino Mega2560	Rs 6000
2	Purchase of 2 x ATMega1281	Rs 4000
3	Shipment Charges for ATMega1281	Rs 8000
4	Construction of Device Circuit	Rs 4000
5	**Total Cost**	**22000**

Appendix 'B'

TIME LINE



Appendix 'C'

Code for reading/writing SD card using Arduino Mega2560

```
#include <SD.h>
File myFile;
void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   Serial.print("Initializing SD card...");
   pinMode(10, OUTPUT);
     if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
  myFile = SD.open("test.txt", FILE_WRITE);
    // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
```

```
    Serial.println("error opening test.txt");
  }

  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");
        // read from the file until there's nothing else in it:
    while (myFile.available()) {
        Serial.write(myFile.read());
    }    // close the file:
    myFile.close();
  } else {
      Serial.println("error opening test.txt");
  }
}

void loop()

{

}
```

Appendix ‘D’

Code for AES implementation

Aes128.h File

```
#ifndef uint8_t
#define uint8_t unsigned char
#endif

#ifdef __cplusplus
extern "C" {
#endif

typedefstruct apple {
uint8_t key[16];
uint8_tenckey[16];
uint8_tdeckey[16];
   } aes128_context;
void aes128_init(aes128_context *, uint8_t *);
void aes128_done(aes128_context *);
void aes128_encrypt_ecb(aes128_context *, uint8_t *);
void aes128_decrypt_ecb(aes128_context *, uint8_t *);
```

```
#ifdef __cplusplus
}
#endif
```

AES.c File

```
#include "aes128.h"

#define F(x)   (((x)<<1) ^ ((((x)>>7) & 1) * 0x1b))
#define FD(x)  (((x) >> 1) ^ (((x) & 1) ? 0x8d : 0))

uint8_t gf_alog(uint8_t x) {
uint8_tatb = 1, z;

while (x--) {z = atb; atb<<= 1; if (z & 0x80) atb^= 0x1b; atb ^= z;}

returnatb;
}
uint8_t gf_log(uint8_t x) {
uint8_tatb = 1, i = 0, z;

do {
if (atb == x) break;
      z = atb; atb<<= 1; if (z & 0x80) atb^= 0x1b; atb ^= z;
   } while (++i> 0);

returni;
}
uint8_t gf_mulinv(uint8_t x)
{
```

```
return (x) ? gf_alog(255 - gf_log(x)) : 0;
}
uint8_t rj_sbox(uint8_t x)
{
uint8_t y, sb;

sb = y = gf_mulinv(x);
   y = (y<<1)|(y>>7); sb ^= y;  y = (y<<1)|(y>>7); sb ^= y;
   y = (y<<1)|(y>>7); sb ^= y;  y = (y<<1)|(y>>7); sb ^= y;

return (sb ^ 0x63);
}

uint8_t rj_sbox_inv(uint8_t x)
{
uint8_t y, sb;

   y = x ^ 0x63;
sb = y = (y<<1)|(y>>7);
   y = (y<<2)|(y>>6); sb ^= y; y = (y<<3)|(y>>5); sb ^= y;

returngf_mulinv(sb);
}

uint8_t rj_xtime(uint8_t x)
{
return (x & 0x80) ? ((x << 1) ^ 0x1b) : (x << 1);
}
voidaes_subBytes(uint8_t *buf)
```

```
{
register uint8_t i = 16;

while (i--) buf[i] = rj_sbox(buf[i]);
}
voidaes_subBytes_inv(uint8_t *buf)
{
register uint8_t i = 16;

while (i--) buf[i] = rj_sbox_inv(buf[i]);
}
voidaes_addRoundKey(uint8_t *buf, uint8_t *key)
{
register uint8_t i = 16;

while (i--) buf[i] ^= key[i];   }
voidaes_addRoundKey_cpy(uint8_t *buf, uint8_t *key, uint8_t *cpk)
{
register uint8_t i = 16;

while (i--)  buf[i] ^= (cpk[i] = key[i]), cpk[16+i] = key[16 + i];
}
voidaes_shiftRows(uint8_t *buf)
{
register uint8_t i, j;
i = buf[1]; buf[1] = buf[5]; buf[5] = buf[9]; buf[9] = buf[13];
buf[13] = i;
i = buf[10]; buf[10] = buf[2]; buf[2] = i;
```

```
   j = buf[3]; buf[3] = buf[15]; buf[15] = buf[11]; buf[11] = buf[7];
buf[7] = j;
   j = buf[14]; buf[14] = buf[6]; buf[6]  = j;

}
voidaes_shiftRows_inv(uint8_t *buf)
{
register uint8_t i, j;
i = buf[1]; buf[1] = buf[13]; buf[13] = buf[9]; buf[9] = buf[5];
buf[5] = i;
i = buf[2]; buf[2] = buf[10]; buf[10] = i;
   j = buf[3]; buf[3] = buf[7]; buf[7] = buf[11]; buf[11] = buf[15];
buf[15] = j;
   j = buf[6]; buf[6] = buf[14]; buf[14] = j;

}

voidaes_mixColumns(uint8_t *buf)
{
register uint8_t i, a, b, c, d, e;

for (i = 0; i< 16; i += 4)
   {
      a = buf[i]; b = buf[i + 1]; c = buf[i + 2]; d = buf[i + 3];
      e = a ^ b ^ c ^ d;
buf[i] ^= e ^ rj_xtime(a^b);   buf[i+1] ^= e ^ rj_xtime(b^c);
buf[i+2] ^= e ^ rj_xtime(c^d); buf[i+3] ^= e ^ rj_xtime(d^a);
   }
}
```

```
voidaes_mixColumns_inv(uint8_t *buf)
{
register uint8_t i, a, b, c, d, e, x, y, z;

for (i = 0; i< 16; i += 4)
   {
      a = buf[i]; b = buf[i + 1]; c = buf[i + 2]; d = buf[i + 3];
      e = a ^ b ^ c ^ d;
      z = rj_xtime(e);
      x = e ^ rj_xtime(rj_xtime(z^a^c));  y = e ^
rj_xtime(rj_xtime(z^b^d));
buf[i] ^= x ^ rj_xtime(a^b);   buf[i+1] ^= y ^ rj_xtime(b^c);
buf[i+2] ^= x ^ rj_xtime(c^d); buf[i+3] ^= y ^ rj_xtime(d^a);
   }
}
voidaes_expandEncKey(uint8_t *k, uint8_t *rc)
{
register uint8_t i;

k[0] ^= rj_sbox(k[29]) ^ (*rc);
k[1] ^= rj_sbox(k[30]);
k[2] ^= rj_sbox(k[31]);
k[3] ^= rj_sbox(k[28]);
   *rc = F( *rc);

for(i = 4; i< 16; i += 4)  k[i] ^= k[i-4],   k[i+1] ^= k[i-3],
k[i+2] ^= k[i-2], k[i+3] ^= k[i-1];
k[16] ^= rj_sbox(k[12]);
k[17] ^= rj_sbox(k[13]);
```

```
k[18] ^= rj_sbox(k[14]);
k[19] ^= rj_sbox(k[15]);

for(i = 20; i< 32; i += 4) k[i] ^= k[i-4],   k[i+1] ^= k[i-3],
k[i+2] ^= k[i-2], k[i+3] ^= k[i-1];

}
voidaes_expandDecKey(uint8_t *k, uint8_t *rc)
{
uint8_ti;

for(i = 28; i> 16; i -= 4) k[i+0] ^= k[i-4], k[i+1] ^= k[i-3],
k[i+2] ^= k[i-2], k[i+3] ^= k[i-1];

k[16] ^= rj_sbox(k[12]);
k[17] ^= rj_sbox(k[13]);
k[18] ^= rj_sbox(k[14]);
k[19] ^= rj_sbox(k[15]);

for(i = 12; i> 0; i -= 4)  k[i+0] ^= k[i-4], k[i+1] ^= k[i-3],
k[i+2] ^= k[i-2], k[i+3] ^= k[i-1];

   *rc = FD(*rc);
k[0] ^= rj_sbox(k[29]) ^ (*rc);
k[1] ^= rj_sbox(k[30]);
k[2] ^= rj_sbox(k[31]);
k[3] ^= rj_sbox(k[28]);
}
void aes128_init(aes128_context *ctx, uint8_t *k)
```

```
{
uint8_trcon = 1;
register uint8_t i;

for (i = 0; i<sizeof(ctx->key); i++) ctx->enckey[i] = ctx->deckey[i]
= k[i];
for (i = 8;--i;) aes_expandEncKey(ctx->deckey, &rcon);
}

void aes128_done(aes128_context *ctx)
{
register uint8_t i;

for (i = 0; i<sizeof(ctx->key); i++)
ctx->key[i] = ctx->enckey[i] = ctx->deckey[i] = 0;
}
void aes128_encrypt_ecb(aes128_context *ctx, uint8_t *buf)
{
uint8_ti, rcon;

aes_addRoundKey_cpy(buf, ctx->enckey, ctx->key);
for(i = 1, rcon = 1; i< 10; ++i)
   {
aes_subBytes(buf);
aes_shiftRows(buf);
aes_mixColumns(buf);
if(i& 1 ) aes_addRoundKey( buf, &ctx->key[16]);
elseaes_expandEncKey(ctx->key, &rcon), aes_addRoundKey(buf, ctx-
>key);
```

```
   }
aes_subBytes(buf);
aes_shiftRows(buf);
aes_expandEncKey(ctx->key, &rcon);
aes_addRoundKey(buf, ctx->key);
}
void aes128_decrypt_ecb(aes128_context *ctx, uint8_t *buf)
{
uint8_ti, rcon;

aes_addRoundKey_cpy(buf, ctx->deckey, ctx->key);
aes_shiftRows_inv(buf);
aes_subBytes_inv(buf);

for (i = 10, rcon = 0x80; --i;)
   {
if( ( i& 1 ) )
      {
aes_expandDecKey(ctx->key, &rcon);
aes_addRoundKey(buf, &ctx->key[16]);
      }
elseaes_addRoundKey(buf, ctx->key);
aes_mixColumns_inv(buf);
aes_shiftRows_inv(buf);
aes_subBytes_inv(buf);
   }
aes_addRoundKey(buf, ctx->key);
}
```

Writing the main function

```
#include<ATMega1281.h>
#include"aes128.h"

_CONFIG2(FNOSC_FRC & FCKSM_CSECMD & POSCMOD_NONE &
OSCIOFNC_ON  )
_CONFIG1(FWDTEN_OFF & JTAGEN_OFF )

uint8_t key[33]="0123456789012345";
uint8_tbuf[]="This program takes 16 bytes of a text and encryptes
it.";
    uint8_t key1[]="KEY....";
    uint8_t text1[]="TEXT:...";
    uint8_t cyphertext1[]="CIPERTEXT...";
    uint8_t decr_text1[]="DECRYPTED TEXT..";
uint8_tm,t;
intz,n,p;
inti,q,w,e,x,j,v,y;
intu,e,w,v,bb,nnn,zz,wx,qq,yy;

void aes128_print(uint8_t *,int,int);
voiden_de_display(uint8_t*,int,int,int);
voidcarriage_return(void);
aes128_contextctx;
int main(void)
{
_COSC2;
_COSC1;
_COSC0;
U1MODEbits.UARTEN=1;
```

```
U1STAbits.UTXEN=1;
        U1MODE = 0; // Clear UART1 mode register
        U1STA = 0;// Clear UART1 status register

        U1BRG = 25;
    IPC2bits.U1RXIP = 4;    //set recieve interrupt priority to 4
    IEC0bits.U1RXIE = 1;    //recieve interrupt enable
    _U1RXIF=0;              // Clear UART RX Interrupt Flag
     U1STAbits.URXISEL = 0;  //interrupt when any character is
recieved
        U1MODEbits.UARTEN = 1;   // Enable UART1 module
        U1STAbits.UTXEN = 1;     // Enable UART1 transmit

Aes128_print(text1,sizeof(text1),0);
carriage_return();
aes128_print(buf,sizeof(buf),0);     //  send the text

carriage_return();
aes128_print(key1,sizeof(key1),0);
carriage_return();

aes128_print(key,sizeof(key),0);  // send the key
carriage_return();
yy=sizeof(buf);

aes128_print(cyphertext1,sizeof(cyphertext1),0);
en_de_display(buf,yy,0,0);      // cipher text

aes128_print(decr_text1,sizeof(decr_text1),0);
```

```
en_de_display(buf,yy,0,1);    //  decrypted text

while(1); //end

return 0;
   }
voiden_de_display(uint8_t* array ,intmm,intconstant,int flag)
{
intz,n,p;
inti,q,w,e,x,j,v;
unsigned char sub[16],b[16];
   n=mm-1;//total n=9
   p=n%16;
carriage_return();

if((p!=0)&&(flag==0))// check if we have exact 16 byte chunks and
encryption required
{
        w=n-p;
     x=0;
for(q=w;q<w+p;q++)
     {
b[x]=array[q];
x++;
               }
for(e=p;e<16;e++)
b[e]='0';      // zeros added
                }
```

```
if(n>=16)
{
j=0;
for( v=0;v<=n-p-1;v++)
                {

sub[j]=array[v];                                    j++;
if(j==16)
                        {
if(flag==0)                     //if encryption required
                         {
                         Aes128_init(&ctx,key);
                         Aes128_encrypt_ecb(&ctx,sub);

Aes128_print(sub,17,constant);
for(qq=0;qq<16;qq++)
array[(v-15+qq)]=sub[qq];       //again shifting encrypted data to
array
      }
if(flag==1)      //if decryption is required
{  // aes128_init(&ctx,key);
                            Aes128_decrypt_ecb(&ctx,sub);
                            Aes128_print(sub,17,constant);

                            }
                          j=0;
                        }
                }
```

```
}
```

Appendix ‘E’

Code for accessing SD Card

Sd.h file

```
#ifndef __SD_H__#define __SD_H__
#include <SdFat.h>
#include <SdFatUtil.h>

#define FILE_READ O_READ
#define FILE_WRITE (O_READ | O_WRITE | O_CREAT)

classFile:public Stream {
 private:
  char _name[13];
  SdFile*_file;
public:
  File(SdFile f, constchar*name);
  File(void);
  virtualsize_twrite(uint8_t);
  virtualsize_twrite(constuint8_t*buf, size_t size);
```

```
  virtualintread();
  virtualintpeek();
  virtualintavailable();
  virtualvoidflush();
  intread(void*buf, uint16_tnbyte);
  booleanseek(uint32_tpos);
  uint32_tposition();
  uint32_tsize();
  voidclose();
  operatorbool();
  char*name();

  booleanisDirectory(void);
  File openNextFile(uint8_t mode = O_RDONLY);
  voidrewindDirectory(void);

  using Print::write;
};

classSDClass {

private:
  Sd2Card card;
  SdVolume volume;
  SdFile root;

  SdFilegetParentDir(constchar*filepath, int*indx);
public:
  booleanbegin(uint8_tcsPin= SD_CHIP_SELECT_PIN);
```

```
  File open(constchar*filename, uint8_t mode = FILE_READ);

  booleanexists(char*filepath);

   booleanmkdir(char*filepath);

  booleanremove(char*filepath);

  booleanrmdir(char*filepath);

private:

  intfileOpenMode;

  friendclassFile;
  friendbooleancallback_openPath(SdFile&, char*, boolean, void*);
};

externSDClass SD;

#endif
```