Comparative Analysis of AES Algorithms and Implementation of AES in Arduino

Thesis report

<u>Supervisor:</u> Dr. Amitabha Chakrabarty

<u>Co-Supervisor:</u> Samiul Islam

<u>Conducted By:</u>

Tanzir Ismat (10101016)

School of Engineering and Computer Science

BRAC University

DECLARATION

I do hereby declare that this thesis titled, **‘Comparative Analysis of AES Algorithms and Implementation of AES in Arduino’** is submitted to the Department Of Compute Science and Engineering of BRAC University in fulfillment of the Bachelor of Science in Computer Science and Engineering. This thesis is based on results found by myself. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole nor in part, has been previously submitted for any degree.

Date: 20.12.2015

Signature of Supervisor	Signature of Author
____________________________	____________________________
Dr. Amitabha Chakrabarty	Tanzir Ismat (10101016)

Thesis Supervisor

Assistant Professor

Department of Computer Science & Engineering

BRAC University.

Acknowledgements

It is an honor for me to thank those who made this thesis possible. I owe my deepest gratitude to my supervisor, Dr. Amitabha Chakrabarty, whose encouragement, guidance and support from the initial level to the end, enabled me to develop an understanding of the subject and helped me to fulfill my thesis work.

This thesis would be not be possible without help from my co-supervisor Samiul Islam. I am very thankful to him and would like to show my gratitude to him for constantly helping and guiding me to complete the work.

I would like to give my thanks to all my faculties of Computer Science & Engineering dept. of BRAC University, my friends and my family for their constant support.

Last but not at least, thanks to the Almighty for helping me in every steps of this Thesis work.

Abstract

The Advanced Encryption Standard (AES) are one of the most significant algorithms used in symmetric key cryptography. Finalist candidate algorithms of AES competition program arranged by National Institute of Standards and Technology (NIST) in 1997 are, five algorithms they are: Rijndael, MARS, RC6, Serpent, and Twofish. From these algorithms Rijndael got the most numbers of votes and selected as the AES algorithm. In this thesis, various finalist candidates of AES algorithms have been analyzed, remarking its main advantages and limitations, memory usage of different algorithms and also the selection criteria of AES finalist algorithms evaluated on various evaluation criteria.Also the aim of this thesis project is to determine if cryptographic software can be implemented in commercially available hardware like Arduino that have limited amount of memory. More specifically, the different amounts of memory (eg. Flash, EEPROM, SRAM) used and remaining need to be determined. For the scope of this project, Advanced Encryption Standard (AES) was considered for the Arduino Mega 2560 platform.

Table of Contents

Chapter 1: Introduction

These days with more and more technological advancements in the field of communication there is even more ever increasing threat to data which is being exposed in the environment of cryptographic attacks. With more advancement in internet applications there is a lot of critical data which is shared by the user and that has to be protected from illegal use by the hackers. As the growth of technology, communication through internet and wireless methods has become a revolutionary advancement of late. So the never changing attribute which is of utmost prominence is the very basic necessity to protect the data from its unauthorized access of the information. With increasing inclination towards information security there was even more predilection in regard to security algorithms which acts as a barricade between the hacker and the critical data. As there were a lot of security algorithms which evolved out of the cause and were gaining its own appreciation at different fields of its use, the US government wanted to standardize a cryptographic algorithm which will be used universally by them called AES (Advanced Encryption Standards).[1] In 1997 NIST announced a program to develop and choose an Advanced Encryption Standard to replace the aging Data Encryption Standard (DES).They solicited algorithms from the cryptographic community, with the intent of choosing a single standard. Fifteen algorithms were submitted to NIST in 1998, and NIST chose five finalists in 1999. In 2000, Rijndael was selected by getting the most votes as the AES algorithm. NIST''s three selection criteria were security, performance, and flexibility. Efficiency of algorithms was also another important criteria. Two characteristics were determined as critical in the selection of the AES: security and efficiency. In this thesis the evaluation and selection criteria of NIST for AES finalist algorithms will be investigated also the comparison between different algorithms will be discussed [2]. Another aim of this thesis project is to determine if cryptographic software can be implemented in commercially available hardware like Arduino that have limited amount of memory. More specifically, the different amounts of memory (eg Flash, EEPROM, SRAM) used and remaining need to be determined. For the scope of this project, Advanced Encryption Standard (AES) was considered for the Arduino Mega 2560 platform.

Before going in to details let us just review few basics.

Literature Survey

While studying for my thesis research works that I found and from where I have taken data to prepare for myself for thesis are mentioned some over here with details and others in the references section.

Comparative Analysis of AES Finalist Algorithms And Low Power Methodology For Rc6 Block Cipher – A review(*by V. Tharun Deep, Dr. Venkata Siva Reddy.)* – I have studied the past works and from here I have learned about different cryptographic algorithms which were shortlisted by NIST (National Institute of Standards and Technology) for the final round of selection for determining the AES also the various evaluation criteria for the cryptographic algorithms and the implementation methodology of the cryptographic algorithm RC6 which contains the detailed briefing of Key Expansion Schedule, Encryption Process, and Decryption process. Also it gives the overview to obtain low power at different stages of synthesis flow.

A Performance Comparison of the Five AES Finalists (*by Bruce Schneier, Doug Whiting),* - From here I have learned performance related topics of five AES finalists algorithms like Software performance of these algorithms on different platform using different language environment.

Implementing Security in a Personal Security Device (*by Priyansha Gupta-UCLA),* - From here I have learnt more about AES algorithm and how to implement AES in Arduino.I have learnt how to setup install AES library in Arduino and also memory usage of Arduino.

Cryptoprocessing on the Arduino (*by Stig Tore Johannesen- NUST)*- From here I have gathered more in-depth knowledge about implementing AES in Arduino.

There are many other publications, online forums and websites from which I have taken information to complete my thesis work; those related links and information about them are given in reference sections .

1.1 Cryptography Basics:

Cryptography is usually referred to as “the study of secret”. Before one can begin to understand cryptography, there are several key concepts that must be understood. Firstly, there are the terms plaintext and cipher text. Plaintext refers to data that is unencrypted while cipher text refers to the data that has been encrypted. Encryption is the process of converting normal text to unreadable form. Decryption is the process of converting encrypted text to normal text in the readable form. There are two main categories of cryptography depending on the type of security keys used to encrypt/decrypt the data. These two categories are: **Asymmetric** and **Symmetric** encryption techniques.

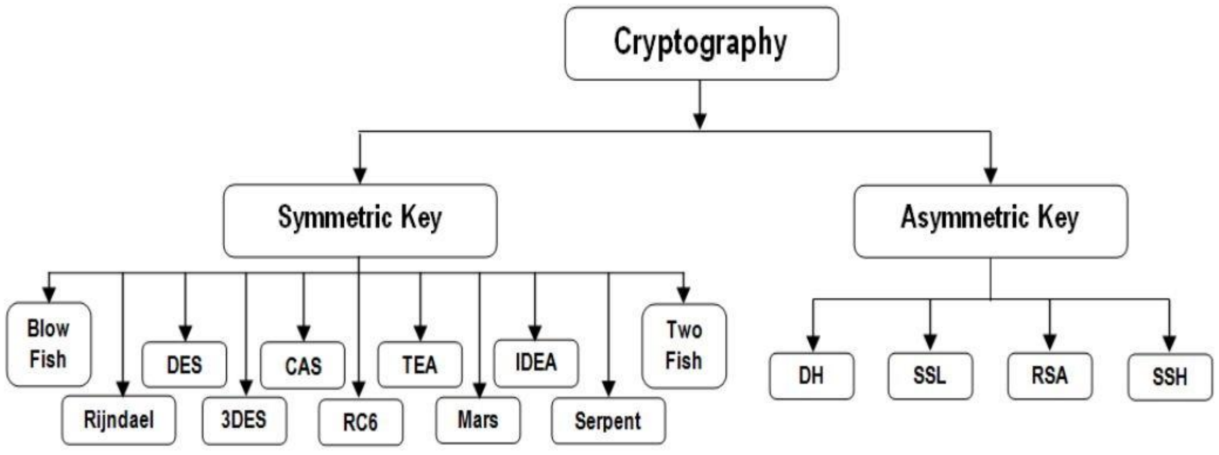


Fig. 1: Different Symmetric and Asymmetric Cryptographic algorithm [3]

A **key** acts as a password that is used to encode and decode data. In the case of symmetric encryption, the key that is used to encrypt the data is the same key that is used to decrypt the data. As a general rule, larger key sizes allow for a larger number of key combinations, which makes it more difficult for an attacker to correctly guess the key, thereby increasing security.

A **round** is another cryptographic term. The number of rounds that an encryption algorithm uses refers to the number of iterations that data is encrypted. The purpose for having more rounds is to achieve a higher level of security, because an increase in rounds translates to an increase in encryption. By increasing the amount of encryption that is done, the resulting cipher text becomes more statistically unrelated to the original plaintext.

Symmetric Encryption:

Symmetry encryption technique is also called as single key cryptography. It uses a single key. In this encryption process the receiver and the sender has to agree upon a single secret (shared) key. Given a message (called plaintext) and the key, encryption produces unintelligible data, which is about the same length as the plaintext was. Decryption is the reverse of encryption, and uses the same key as encryption. Rijndael(AES), MARS, RC6, Serpent, and Twofish these are most used symmetric algorithms.

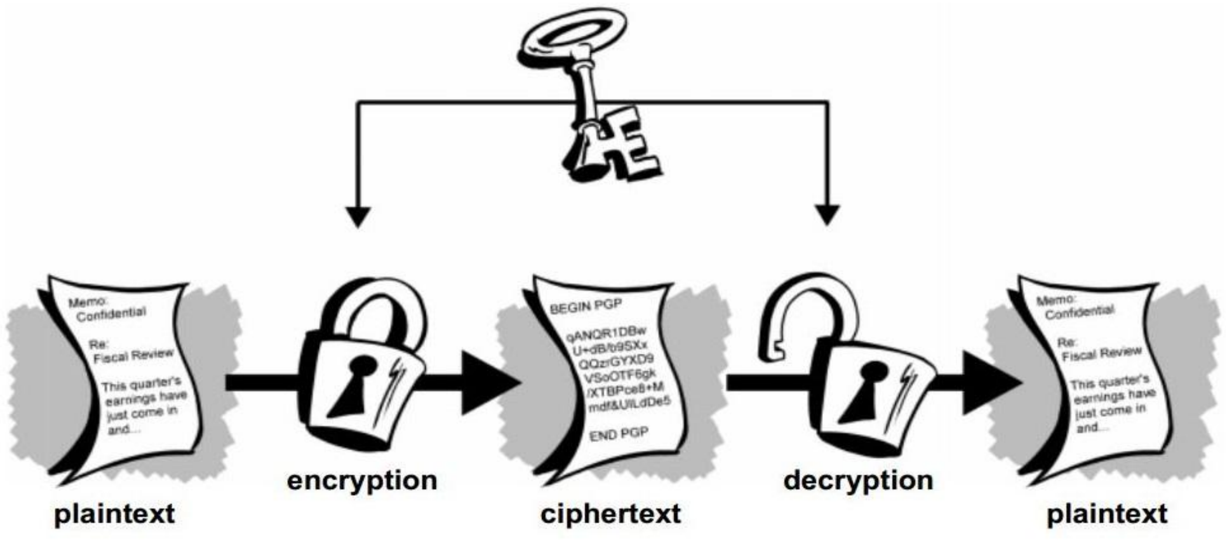

Memo: Confidential
Re: Fiscal Review
This quarter's earnings have just come in and...
encryption
BEGIN PGP
qANQR1DBw U+dB/b9SXx QQz/GYXD9 VSoOTF6gk /XTBPce8+M mdf&UlLdDe5
END PGP
decryption
plaintext
ciphertext
plaintext

Fig.2 Symmetric algorithm encryption and decryption process

1.2 AES finalist algorithms:

The US government wanted to standardize a cryptographic algorithm which will be used universally by them called AES (Advanced Encryption Standards).In 1997 NIST announced a program to develop and choose an Advanced Encryption Standard to replace the aging Data Encryption Standard (DES). They solicited algorithms from the cryptographic community, with the intent of choosing a single standard. Fifteen algorithms were submitted to NIST in 1998, and NIST chose five finalists in 1999.These are Rijndael(AES), MARS, RC6, Serpent, and Twofish.

1.2.1Evaluation Criteria and Final Score of AES Finalist Algorithms:

NIST focused their evaluation of each algorithm based on the following criteria. In order of their stated importance, they were:

1. Security (the most important factor in the evaluation)
2. Cost
3. Algorithm and Implementation Characteristics.

On the basis of these criteria NIST chose the best algorithm and after voting and the final score-

Criteria	Rijndael	Serpent	Twofish	Mars	RC6
General Security	**2**	**3**	**3**	**3**	**2**
Implementation Difficulty	**3**	**3**	**2**	**1**	**1**
Software performance	**3**	**1**	**1**	**2**	**2**
Smart Card Performance	**3**	**3**	**2**	**1**	**1**
Hardware Performance	**3**	**3**	**2**	**1**	**2**
Design Features	**2**	**1**	**3**	**2**	**1**
Total	**16**	**14**	**13**	**10**	**09**

Table 1: Final Score of AES Finalist Algorithms [4]

From the table above we can see that, Rijndael got the most numbers of votes and selected as the AES.

1.2.1 Comparison between different AES Finalist Algorithms:

Architectural Comparison:

Based on the Architecture of these shortlisted algorithms comparison can be summed up through a table as shown below-

Algorithms	**Type of Structure**	**Key Length**	**Block Size With number of rounds**	**S-Boxes**
Rijndael	Feistel Structure	Variable 128, 192 or 256 bits	128 bit With variable 10, 12 or 14 rounds	No
Twofish	Fiestel structure	Variable 128, 192 or 256 bits	128 bit with 16 rounds	Four
Serpent	Substitution permutation network structure	Variable 128, 192 or 256 bits	128 bit with 32 rounds	Eight
MARS	Heterogeneous structure	Variable 128 to 448 bits in multiples of 32-bit	128 bit with 32 rounds	One
RC6	Feistel Structure	Variable 128, 192 or 256 bits	128 bit with 20 rounds.	No

Table 2: Architectural comparison of different AES finalist algorithms [5]

1.2.2 Comparison of different AES finalist Algorithms on the basis of Memory Usage

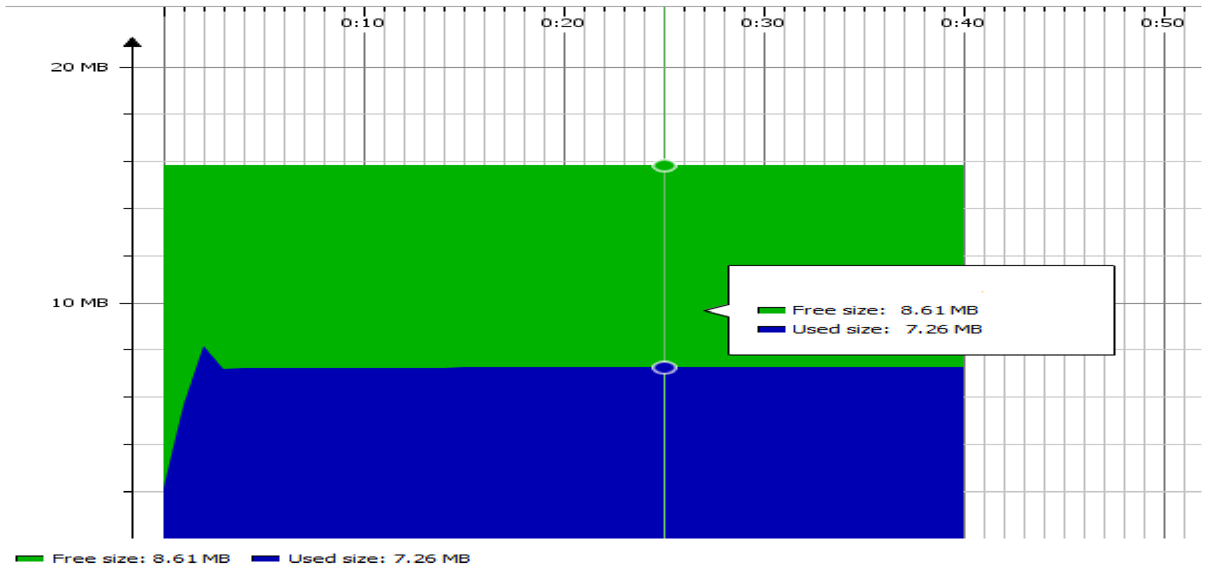
The memory usage can be defined as the number of functions performed by the algorithm, smaller the memory usage greater will be the efficiency. So it’s really an important issue in terms of efficiency of AES algorithms. To calculate the memory usage of different AES algorithms, I have done the following process-

- Implementing of different AES finalist algorithms on laptop Pc
  (Configuration: Intel Core 2 Duo 2.2 Ghz processor, RAM: 3GB)
- Medium of Language used: Java
- IDE used: Eclipse MARS 4.5
- Method of calculation of Memory Usage: I have used a profiler called Jprofiler which is used specially for monitoring memory usage and leaks in any Java applications. This profiler can be download from here- http://www.ej-technologies.com/download/jprofiler/files.
- To calculate the memory, in this calculation steps heap memory is considered.

Memory Usage of different algorithms: After implementing different AES finalist algorithms in Eclipse and integrating Eclipse IDE with Jprofiler I have got the results of the heap memory usages of different algorithms from the Jprofiler application.The result of these memory usage calculations is given below –

1) AES (Rijndael):

Memory

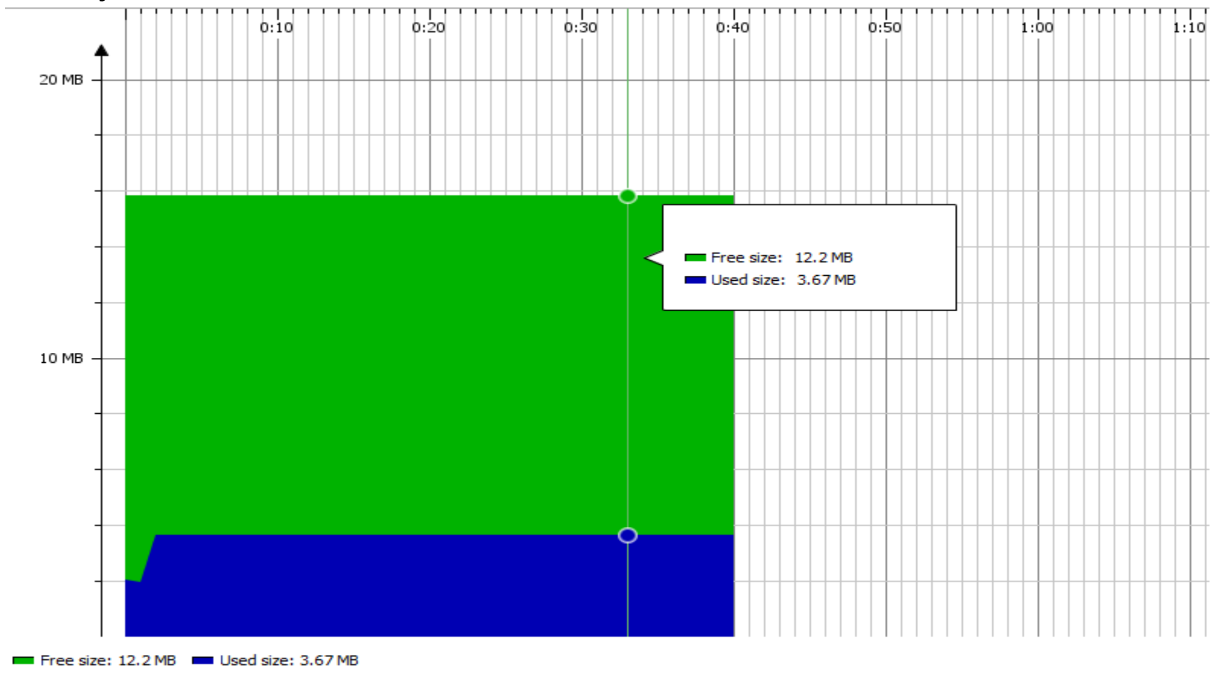


Processing time

Fig 3: Memory usage of AES (Rijndael)

2) Twofish:

Memory

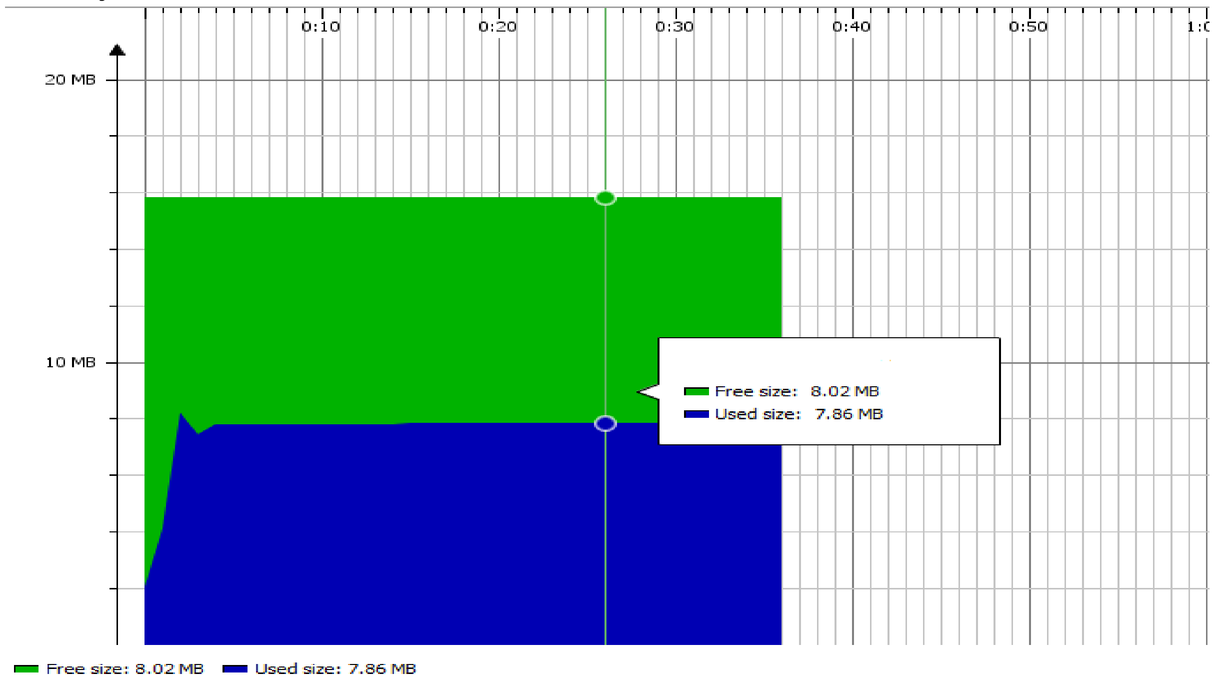


Processing Time

Fig:4 Memory usage of Twofish

3)Serpent:

Memory

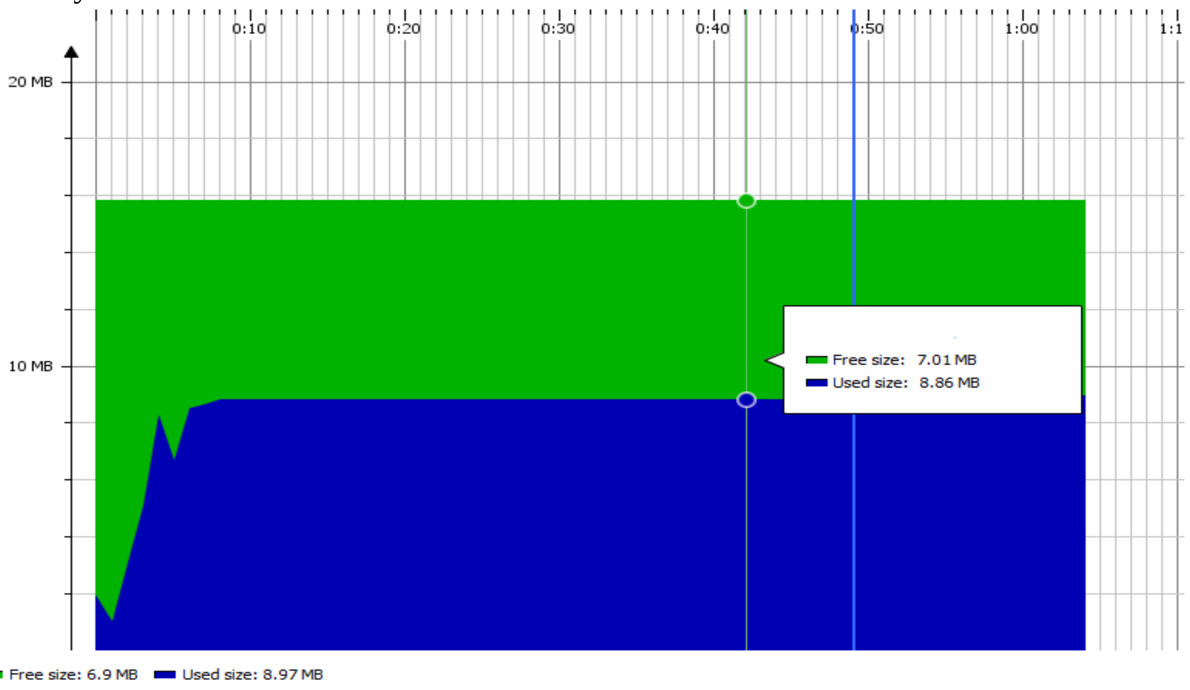


Processing Time

Fig:5 Memory usage of Serpent

4)MARS:

Memory

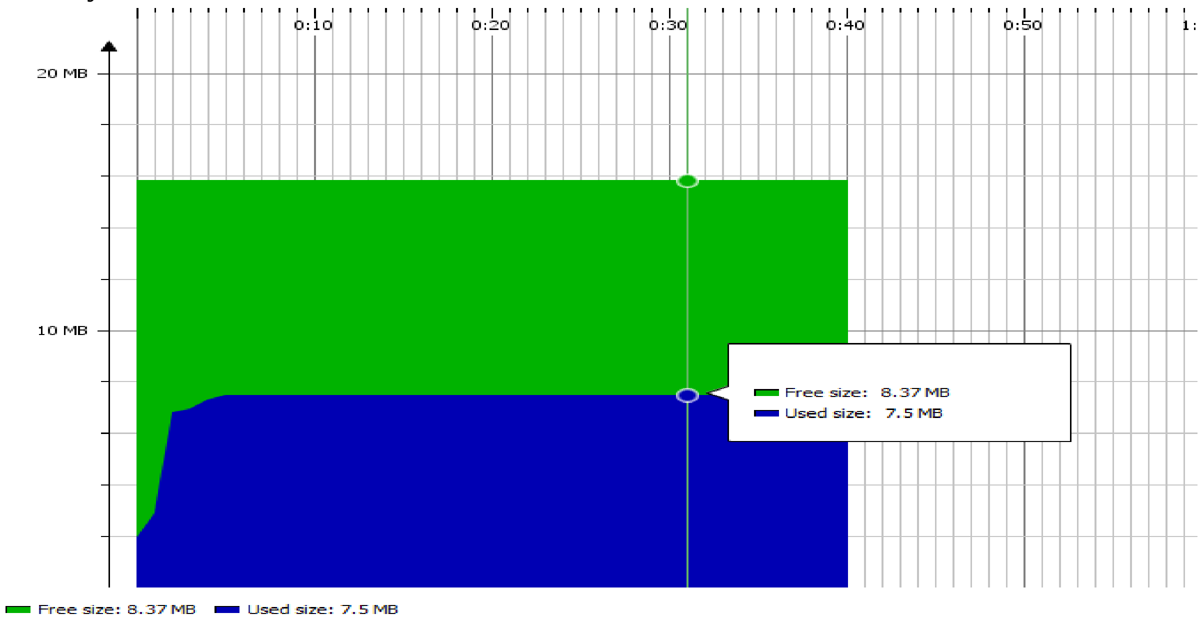


Processing time

Fig: 6 Memory usage of MARS

3) RC6

Memory



Processing Time

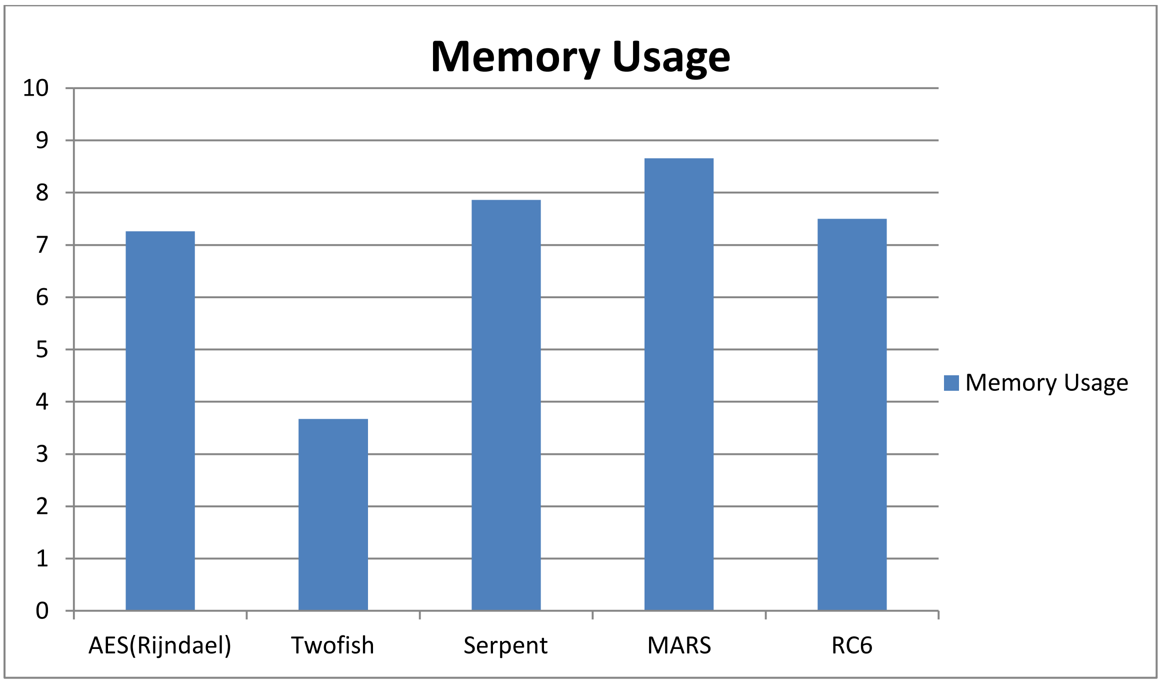
Fig : 7 Memory usage of RC6

Discussions regarding the memory usage results:

After observing the of memory usage results of AES finalist algorithms, it is seen that MARS uses highest memory and also the processing time is higher than the other algorithm. And on the other hand, Twofish uses lowest memory. Here one thing should be considered that the calculation also depends on implementations of these algorithms. Different implementations of these algorithms can bring different memory usage results.

Summary of the Memory Usage of different algorithms:

Memory(mb)



Different AES finalist algorithms

Fig: 8 Memory usage of different AES finalist algorithms

Chapter 2: The Advanced Encryption Standard (AES):

This section will cover, in brief, the Advanced Encryption Standard. (The entire section is a part of a chapter taught at Purdue. See reference [6]).

2.1 Salient feature of AES):

AES is a block cipher with a block length of 128 bits. It allows for three different key lengths: 128, 192, or 256 bits. AES Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Except for the last round in each case, all other rounds are identical. Each round of processing includes one single-byte based substitution step, a row-wise permutation step, a column-wise mixing step, and the addition of the round key. The order in which these four steps are executed is different for encryption and decryption. To appreciate the processing steps used in a single round, it is best to think of a 128-bit×4 block matrixes consisting of a 4
of bytes, arranged as follows:

$$\left[\begin{array}{lllll} \text{byte0} & \text{byte4} & \text{byte8} & \text{byte12} & \text{byte1} \\ \text{byte5} & \text{byte9} & \text{byte13} & \text{byte2} & \text{byte6} \\ \text{byte10} & \text{byte14} & \text{byte3} & \text{byte7} & \\ \text{byte11} & \text{byte15} & & & \end{array}\right]$$

Therefore, the first four bytes of a 128-bit input block occupy the first× column in the 4 4 matrix of bytes. The next four bytes occupy the second column, and so on. The×4 4 matrix of bytes is referred to as the state array. Each round of processing works on the input state array and produces an output state array. The output state array produced by the last round is rearranged into a 128-bit output block. Unlike DES, the decryption algorithm differs substantially from the encryption algorithm. Although, overall, the same

steps are used in encryption and decryption, the order in which the steps are carried out is different. Whereas AES requires the block size to be 128 bits, the original Rijndael cipher works with any block size (and any key size) that is a multiple of 32 as long as it exceeds 128. The state array for the different block sizes still has only four rows in the Rijndael cipher. However, the number of columns depends on size of the block. For example, when the block size is 192, the Rijndael cipher requires a state array to consist of 4 rows and 6 columns.

AES uses a substitution-permutation network in a more general sense. Each round of processing in AES involves byte-level substitutions followed by word-level permutations. The nature of substitutions and permutations in AES allows for a fast software implementation of the algorithm.

2.2 Working and Structure of AES:

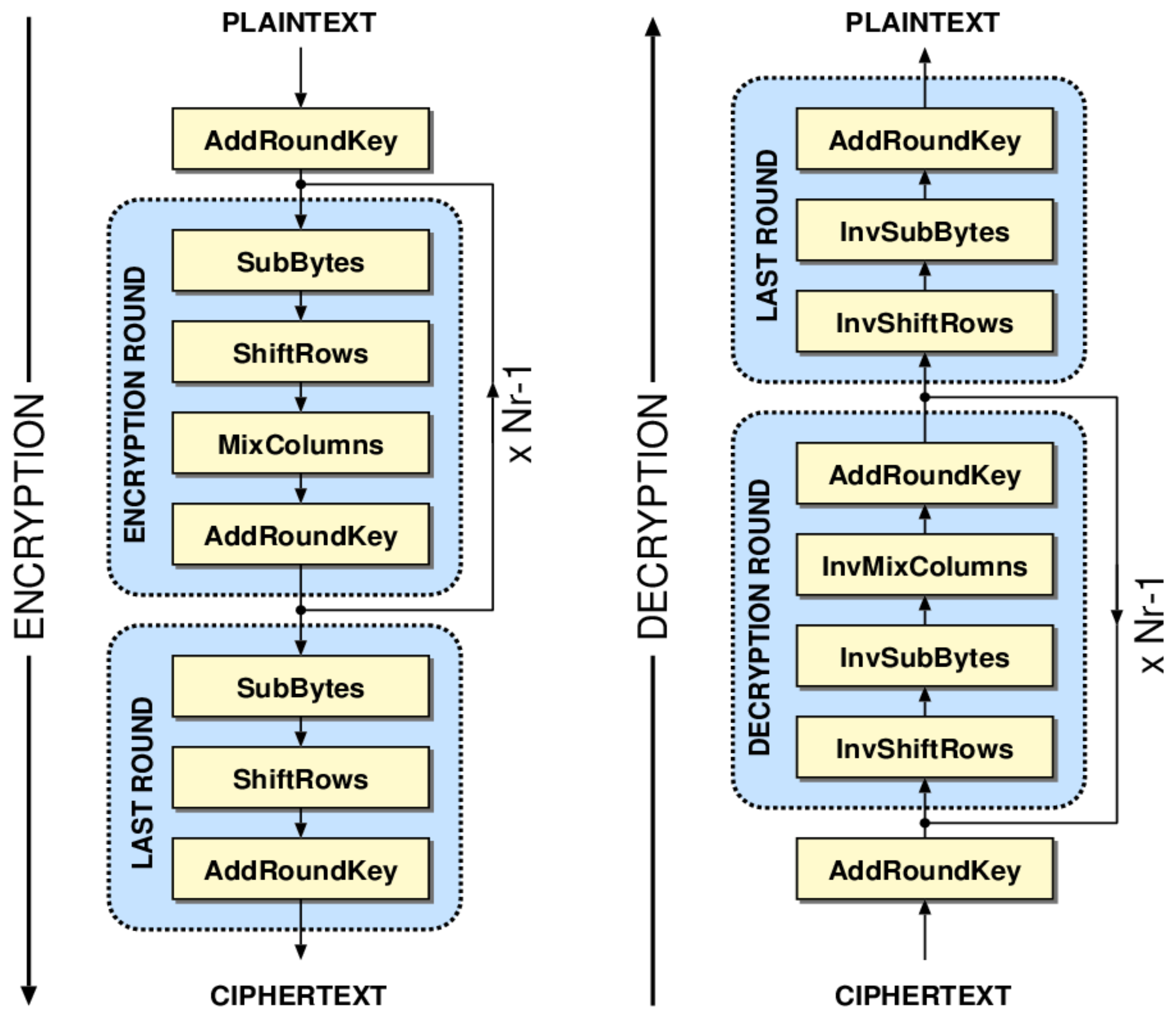


Fig 9: AES encryption and decryption process

Before any round-based processing for encryption can begin, the input state array is XORed with the first four words of the key schedule. The same thing happens during decryption — except that now we XOR the cipher text state array with the last four words of the key schedule. For encryption, each round consists of the following four steps:

I. Substitute bytes

II. Shift rows

III. Mix columns IV. Add
round key.

The last step consists of XORing the output of the previous three steps with four words from the key schedule. For decryption, each round consists of the following four steps:

I. Inverse shift rows

II. Inverse substitute bytes

III. Add round key

IV. Inverse mix columns.

The third step consists of XORing the output of the previous two steps with four words from the key schedule. Note the differences between the order in which substitution and shifting operations are carried out in a decryption round vis-a-vis the order in which similar operations are carried out in an encryption round. The last round for encryption does not involve the “Mix columns” step. The last round for decryption does not involve the “Inverse mix columns” step [5].

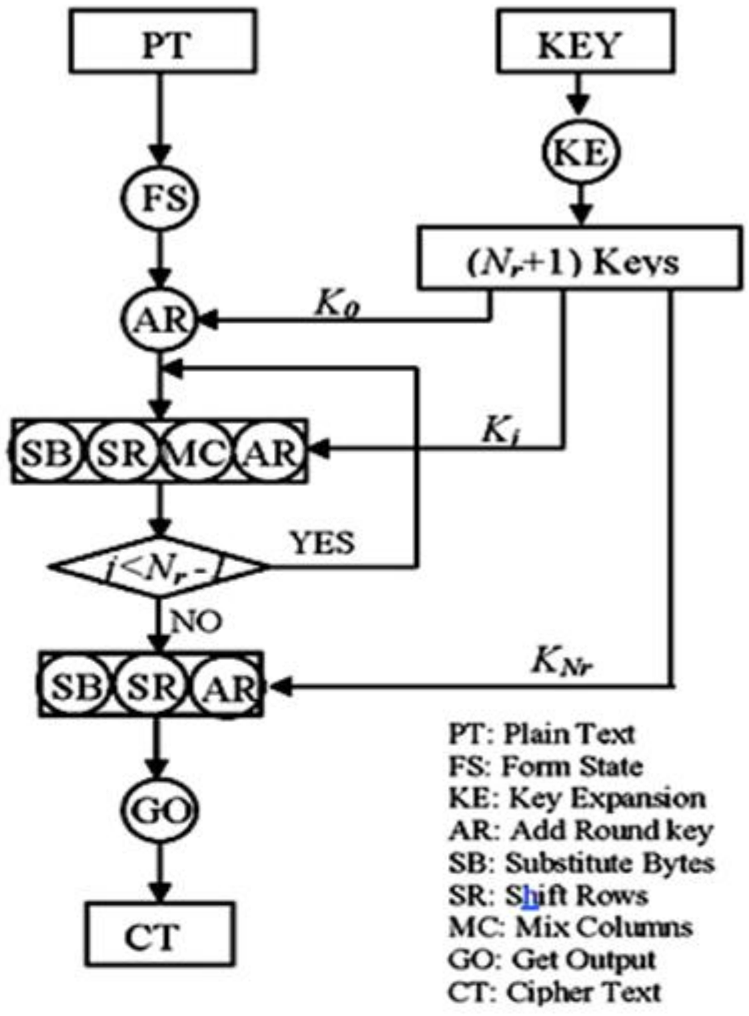


Fig 10: Flow chart of AES encryption and decryption process

In terms of security no attack has been found for AES, using the full number of rounds, which is more efficient than 296 for AES-192. Of specific note in this context is the fact that AES, due to a large avalanche effect, is considered immune to related text attacks. The avalanche effect in this context referring to the property that a small differences in in-put leads to large differences in the output. However some side-channel attacks, attacks on implementations that leak information in some way, has been found for several implementations of AES, though most of them have been quickly patched when the attack became known.

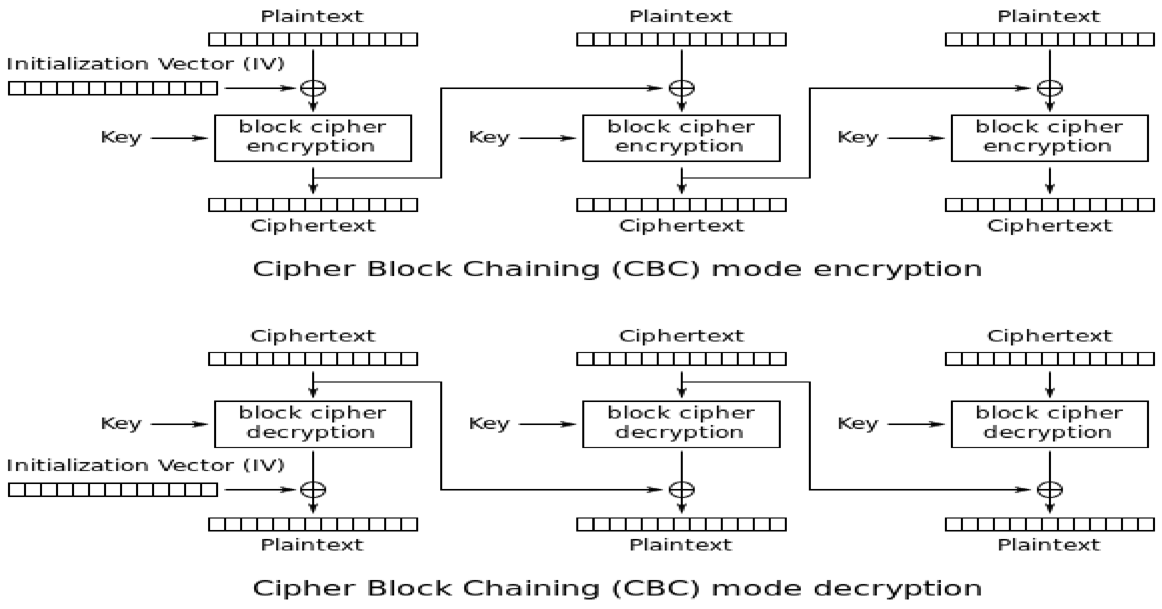
2.3 Modes of operation:

2.3.1 Electronic Code Book (ECB)

Electronic Code Book mode is the simplest of all the modes, as it is simply a lack of any additional operations on top of the encryption algorithm. It simply calculates each block. independently of each other block. While this means the mode is highly parallelizable, it has severe security implications. As long as the encryption key and algorithms are safe it is still impossible to directly decrypt the message, but due to a small block size (128 bits) it is possible to see patterns in the output. If you know the input for some of the output it is also possible to know the input of a given output. Figure 3.1 demonstrates this quite well by showing the ouput of ECB and CBC mode next to the original input, with a clearly visible pattern in the output of ECB mode.

2.3.2 Cipher Block Chaining (CBC)

Cipher Block Chaining is among the simpler true encryption modes of operation. It works by XORing the cipher text of the previous block with the plaintext of the current block when encrypting, and with the result of the decryption operation when decrypting. For the first block an Initialization Vector is used. Provided the IV is unique this mode is considered secure. It does limit the operations to being serial, since the result of the previous operation needs to be known before the current operation can be processed.


Plaintext
Initialization Vector (IV)
Key
block cipher encryption
Ciphertext
Cipher Block Chaining (CBC) mode encryption
Ciphertext
Key
block cipher decryption
Initialization Vector (IV)
Plaintext
Cipher Block Chaining (CBC) mode decryption

Courtesy of Wikimedia Commons

Fig 11: CBC Encryption and Decryption

Chapter 3. Implementation of AES in Arduino

There are three steps involved to implement AES in Arduino and checking the memory usage. These are-
1. Installing AES library on Arduino.
2. Running codes to check the functionalities of AES library.
3. Checking the amount of memory used and remaining.

3.1 Hardware :

The descriptions of the hardware that are implemented in this project are given below:

- Arduino Mega 2560 and
- A laptop Computer/ Desktop PC for giving the command.

The development kit used for this system is an Arduino brand kit. The Arduino family of kits was chosen because it is the most popular and easily available, and because it has both an established development environment and an active community.

3.1.1 Arduino Mega 2560 : (The entire section was taken from official Arduino website: See reference [7])

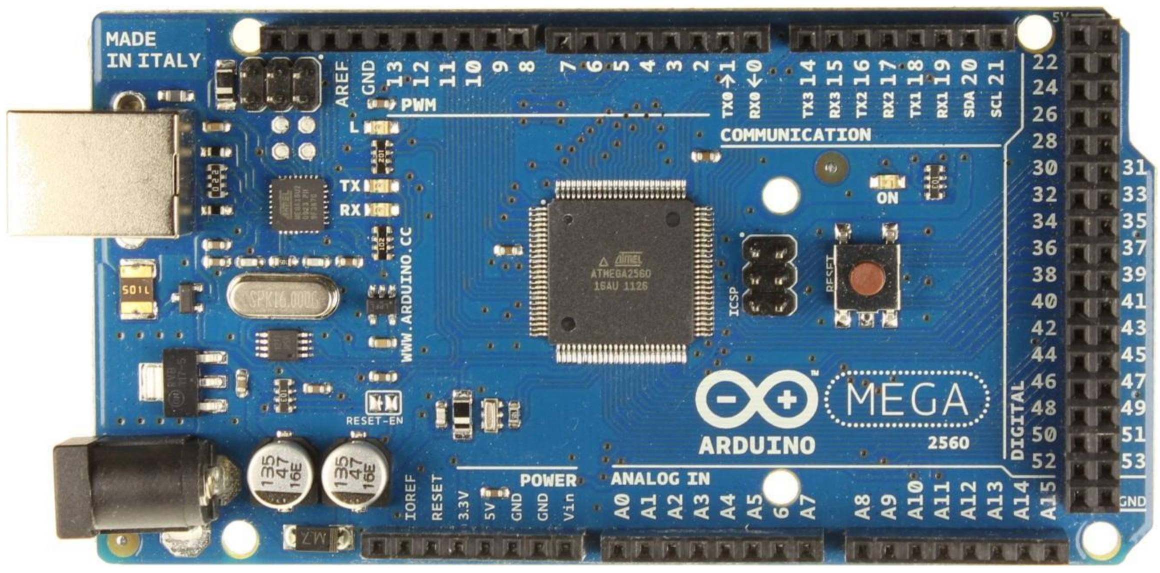


Fig 12: Arduino Mega 2560 microcontroller board

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 . It has 54 digital input/output pins (of which 15 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. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila [6].

A. ARDUINO MEGA 2560 SPECIFICATION

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by boot loader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

B. POWER

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

C. MEMORY

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

D. COMMUNICATION

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega16U2 (ATmega 8U2 on the revision 1 and revision 2 boards) on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a.inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A Software Serial library allows for serial communication on any of the Mega2560's digital pins. The ATmega2560 also supports TWI and SPI communication. The Arduino 6 software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

E. PROGRAMMING

The Arduino Mega can be programmed with the Arduino software. The ATmega2560 on the Arduino Mega comes preburned with a boot loader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol.

3.1.2 Types of memory in an Arduino device :

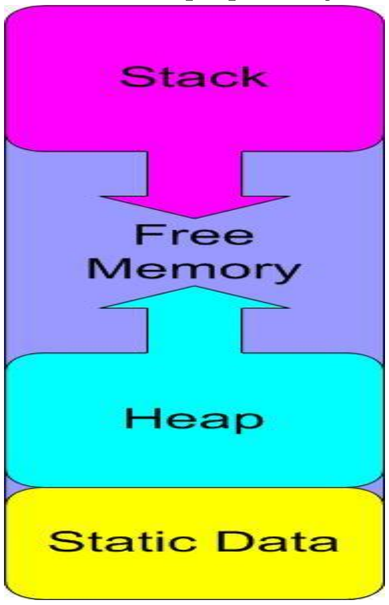
There are three types of memory in an Arduino:

A) FLASH MEMORY:

Flash memory is used to store program image and any initialized data. The flash is usually used to hold the executables (and perhaps other static data) for the device. It is possible to execute program code from flash, but one can't modify data in flash memory from your executing code. To modify the data, it must first be copied into SRAM. Flash memory has a finite lifetime of about 100,000 write cycles. So if 10 programs are uploaded 10 a day, every day for the next 27 years, one might wear it out

B) SRAM

SRAM or **Static Random Access Memory**, can be read and written from executing program. This is where temporary variables are stored. SRAM memory is used for several purposes by a running program:



- **Static Data** - This is a block of reserved space in SRAM for all the global and static variables from program. For variables with initial values, the runtime system copies the initial value from Flash when the program starts.
- **Heap** - The heap is for dynamically allocated data items. The heap grows from the top of the static data area up as data items are allocated.

•**Stack** - The stack is for local variables and for maintaining a record of interrupts and function calls. The stack grows from the top of memory down towards the heap. Every interrupt, function call and/or local variable allocation causes the stack to grow. Returning from an interrupt or function call will reclaim all stack space used by that interrupt or function.

Most memory problems occur when the stack and the heap collide. When this happens, one or both of these memory areas will be corrupted with unpredictable results. In some cases it will cause an immediate crash. In others, the effects of the corruption may not be noticed until much later.

C) EEPROM

It can only be read byte-by-byte, so it can be a little awkward to use. The EEPROM is used to store long-term information developed during the device's use. It is also slower than SRAM and has a finite lifetime of about 100,000 write cycles (you can read it as many times as you want). While it can't take the place of precious SRAM, there are times when it can be very useful! [6]

3.2 Issues with implementing AES in Arduino:

A major concern using Arduino is the limited memory available. The difference between the Arduino microcontrollers and a general purpose computer is the sheer amount of memory available. The Arduino we are using has only 256K bytes of Flash memory, 8K bytes of SRAM and 4K bytes of EEPROM. That is 100,000 times LESS physical memory than a low-end PC! And that's not even counting the disk drive!

As described above, AES encryption requires 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Each round of processing includes one single-byte based substitution step, a row-wise permutation step, a column-wise mixing step, and the addition of the round key. The process of encryption will require some memory space to store temporary results and the final encrypted results which go in flash. Key will be in the EEPROM. The data to be encrypted, any intermediate temporary results, and the encrypted block would probably go in SRAM We need to determine if we can fit reasonable cryptographic software (probably AES) on this device, while still leaving room for other functionality. Alternatively, if we use AES crypto, how much of our space will be available for other operations.

3.2.1 Solution of the problems:

Working in this minimalist environment, resources should be used wisely.

A) Installing AES on Arduino

First step towards solving the problem was to download the Arduino integrated development environment (IDE) software which is an open source and is available on Arduino's official website. Arduino IDE 1.0.5 is downloaded for this project. The next step was to install AES on Arduino.

B) AES library for Arduino

With some research, I was able to find an AES library that supports 128, 192 and 256 bit key sizes. This library can be found here: http://utter.chaos.org.uk:/~markt/AES-library.zip.

C) Running codes to check the functionalities of AES library

For checking the functionalities of AES library I have took some ideas from the AES algorithm I've found from here- http://www.cs.utsa.edu/~wagner/laws/AESintro.html Here is the AES algorithm is outline form, using Java syntax for the pseudo-code, and much of the AES standard notation:

Outline of the AES Algorithm:

```
Constants: int Nb = 4; //
           int Nr = 10, 12, or 14; // rounds, for Nk = 4, 6, or 8
Inputs: array in  of 4*Nb bytes // input plaintext
        array out of 4*Nb bytes // output ciphertext
        array w of 4*Nb*(Nr+1) bytes // expanded key
Internal work array:
       state, 2-dim array of 4*Nb bytes, 4 rows and Nb cols
  Algorithm:
       void Cipher(byte[] in, byte[] out, byte[] w) {
            byte[][] state = new byte[4][Nb];
            state = in; // actual component-wise copy
            AddRoundKey(state, w, 0, Nb - 1);
            for (int round = 1; round < Nr; round++) {
               SubBytes(state);
               ShiftRows(state);
               MixColumns(state);
               AddRoundKey(state, w,
                   round*Nb, (round+1)*Nb - 1);
            }
            SubBytes(state);
            ShiftRows(state);
            AddRoundKey(state, w, Nr*Nb, (Nr+1)*Nb - 1);
            out = state; // component-wise copy
         }
```

3.2.2 Output of Encryption and Decryption code: After running the code for encryption and decryption in Arduino IDE based on the idea from the above mentioned algorithm to produce the following output:

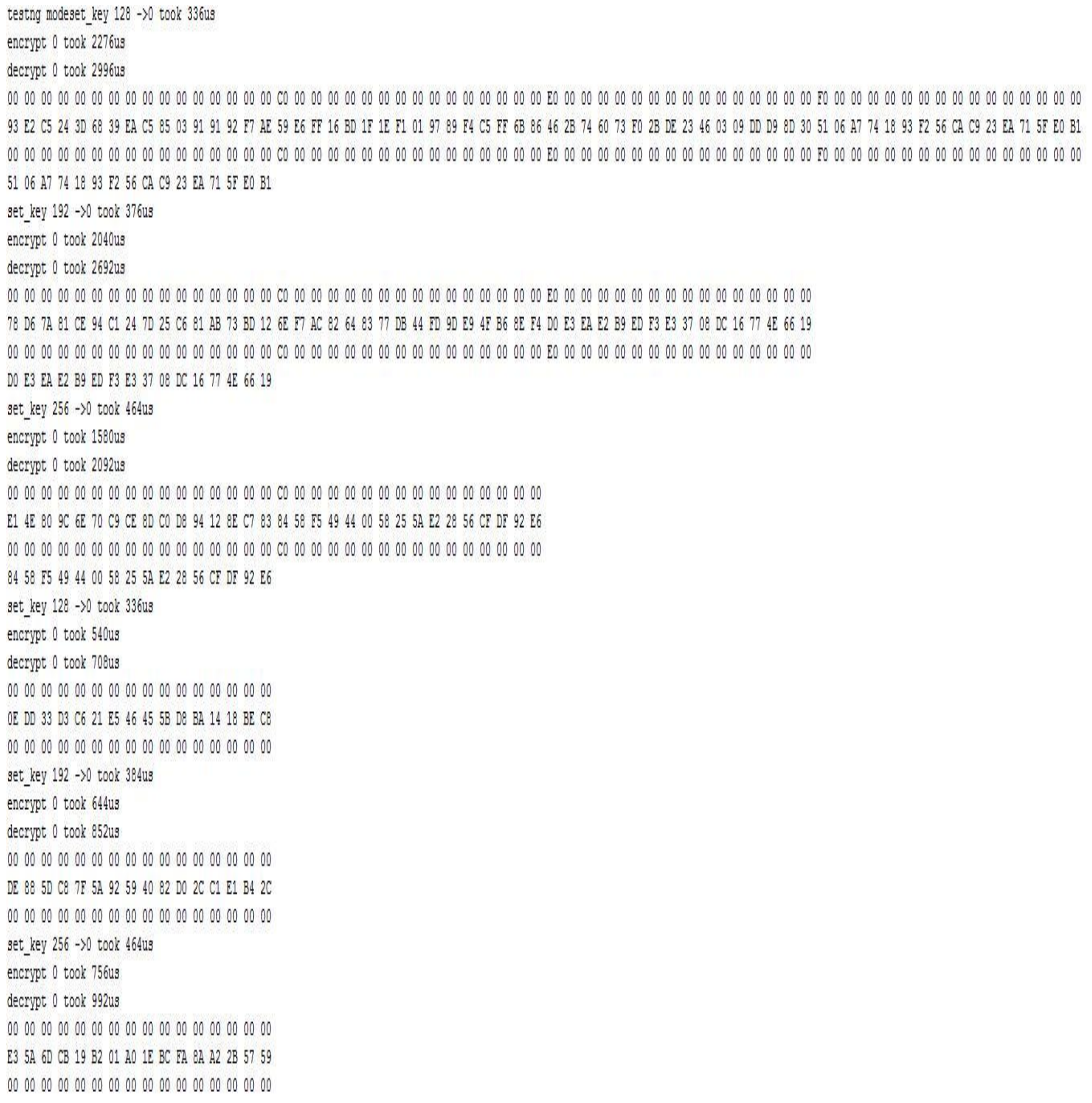
I. Plain text

II. Encrypted text (with varying block size)

III. Decrypted text (with varying block size)

IV. Encryption and decryption using 128, 192 and 256 bit key size

V. Time taken by each and every encryption and decryption process



```
testng modeset_key 128 ->0 took 336us
encrypt 0 took 2276us
decrypt 0 took 2996us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93 E2 C5 24 3D 68 39 EA C5 85 03 91 91 92 F7 AE 59 E6 FF 16 BD 1F 1E F1 01 97 89 F4 C5 FF 6B 86 46 2B 74 60 73 F0 2B DE 23 46 03 09 DD D9 8D 30 51 06 A7 74 18 93 F2 56 CA C9 23 EA 71 5F E0 B1
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51 06 A7 74 18 93 F2 56 CA C9 23 EA 71 5F E0 B1
set_key 192 ->0 took 376us
encrypt 0 took 2040us
decrypt 0 took 2692us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
78 D6 7A 81 CE 94 C1 24 7D 25 C6 81 AB 73 BD 12 6E F7 AC 82 64 83 77 DB 44 FD 9D E9 4F B6 8E F4 D0 E3 EA E2 B9 ED F3 E3 37 08 DC 16 77 4E 66 19
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0 E3 EA E2 B9 ED F3 E3 37 08 DC 16 77 4E 66 19
set_key 256 ->0 took 464us
encrypt 0 took 1580us
decrypt 0 took 2092us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E1 4E 80 9C 6E 70 C9 CE 8D C0 D8 94 12 8E C7 83 84 58 F5 49 44 00 58 25 5A E2 28 56 CF DF 92 E6
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
84 58 F5 49 44 00 58 25 5A E2 28 56 CF DF 92 E6
set_key 128 ->0 took 336us
encrypt 0 took 540us
decrypt 0 took 708us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E DD 33 D3 C6 21 E5 46 45 5B D8 BA 14 18 BE C8
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
set_key 192 ->0 took 384us
encrypt 0 took 644us
decrypt 0 took 852us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
DE 88 5D C8 7F 5A 92 59 40 82 D0 2C C1 E1 B4 2C
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
set_key 256 ->0 took 464us
encrypt 0 took 756us
decrypt 0 took 992us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E3 5A 6D CB 19 B2 01 A0 1E BC FA 8A A2 2B 57 59
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

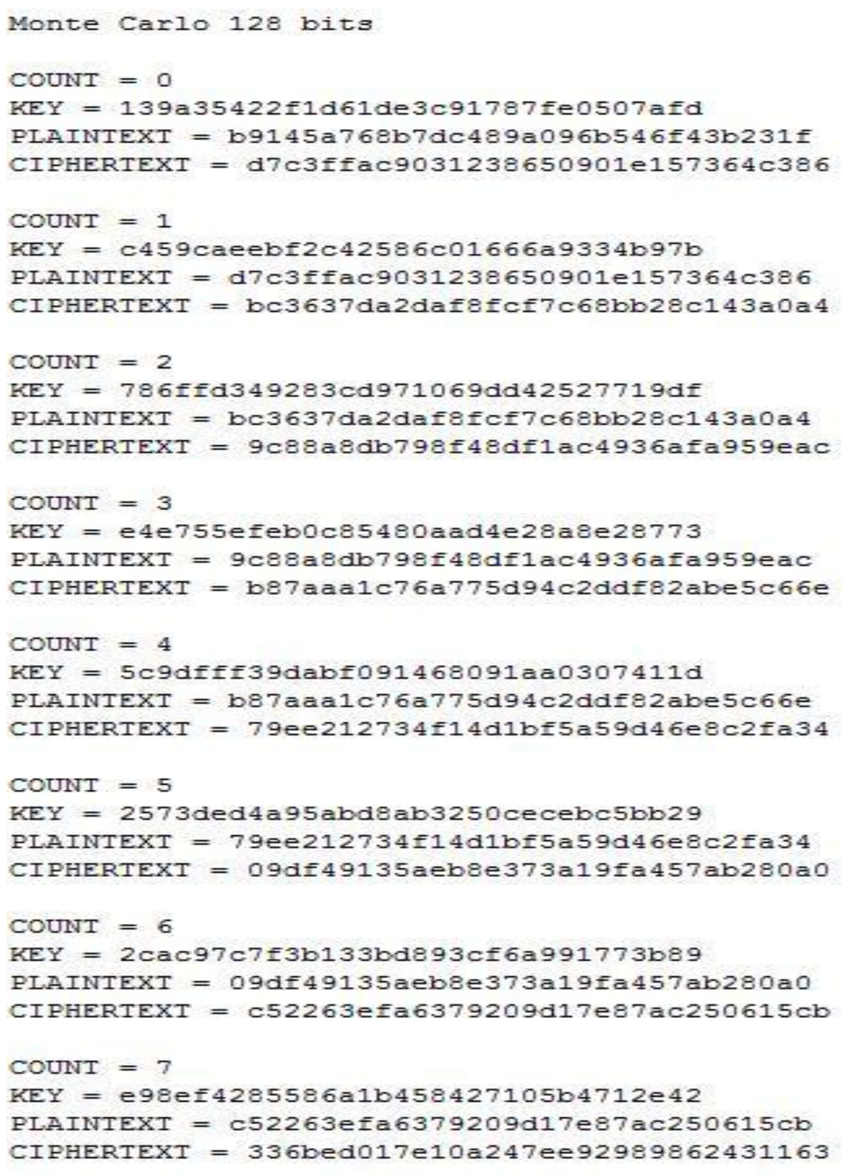
Fig 13: Output showing encryption and decryption process and the amount of time it took

3.2.3 Checking test vectors: The output for Checking test vectors code is showing the following scenarios:

I. Varying size key (128, 192 and 256)

II. Varying plain text and its cipher text

III. Monte Carlo



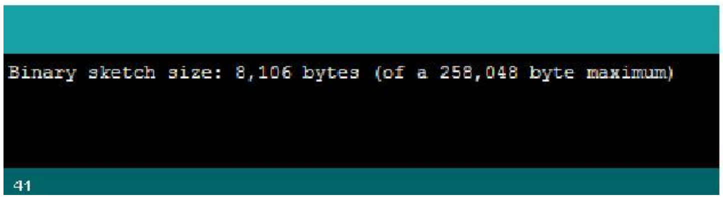
```
Monte Carlo 128 bits

COUNT = 0
KEY = 139a35422f1d61de3c91787fe0507afd
PLAINTEXT = b9145a768b7dc489a096b546f43b231f
CIPHERTEXT = d7c3ffac9031238650901e157364c386

COUNT = 1
KEY = c459caeebf2c42586c01666a9334b97b
PLAINTEXT = d7c3ffac9031238650901e157364c386
CIPHERTEXT = bc3637da2daf8fcf7c68bb28c143a0a4

COUNT = 2
KEY = 786ffd349283cd971069dd42527719df
PLAINTEXT = bc3637da2daf8fcf7c68bb28c143a0a4
CIPHERTEXT = 9c88a8db798f48df1ac4936afa959eac

COUNT = 3
KEY = e4e755efeb0c85480aad4e28a8e28773
PLAINTEXT = 9c88a8db798f48df1ac4936afa959eac
CIPHERTEXT = b87aaa1c76a775d94c2ddf82abe5c66e

COUNT = 4
KEY = 5c9dfff39dabf091468091aa0307411d
PLAINTEXT = b87aaa1c76a775d94c2ddf82abe5c66e
CIPHERTEXT = 79ee212734f14d1bf5a59d46e8c2fa34

COUNT = 5
KEY = 2573ded4a95abd8ab3250cecebc5bb29
PLAINTEXT = 79ee212734f14d1bf5a59d46e8c2fa34
CIPHERTEXT = 09df49135aeb8e373a19fa457ab280a0

COUNT = 6
KEY = 2cac97c7f3b133bd893cf6a991773b89
PLAINTEXT = 09df49135aeb8e373a19fa457ab280a0
CIPHERTEXT = c52263efa6379209d17e87ac250615cb

COUNT = 7
KEY = e98ef4285586a1b458427105b4712e42
PLAINTEXT = c52263efa6379209d17e87ac250615cb
CIPHERTEXT = 336bed017e10a247ee9298986243l163
```

Fig 14: Test vector code output

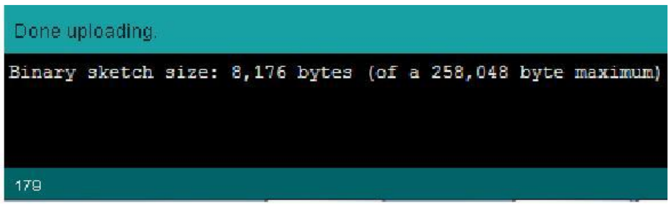
3.2.4 Checking the amount of memory used and remaining:

1) Flash

The amount of flash memory used can be found out at the bottom part of the sketch. One can see the amount of bytes being used after uploading the program. Both encryption & decryption and test vector code was taking approximately 8000 bytes out of total 258,048 bytes in Arduino.



```
Binary sketch size: 8,106 bytes (of a 258,048 byte maximum)
```



```
Done uploading.
Binary sketch size: 8,176 bytes (of a 258,048 byte maximum)
```

Fig 15: Flash memory used by encryption

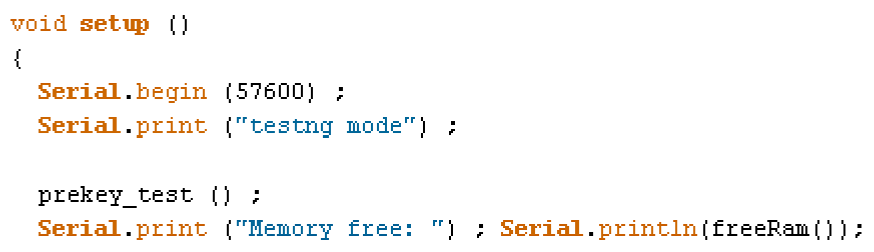
Fig 16: Flash memory used by test vectors

2) EEPROM

EEPROM usage is in fully controlled by user, i.e. we have to read and write each byte to a specific address. So if we want to save 128 bit key in the EEPROM, it will take only 16 bytes in EEPROM. Similarly, a 192 and 256 key will take 24 and 32 bytes respectively.

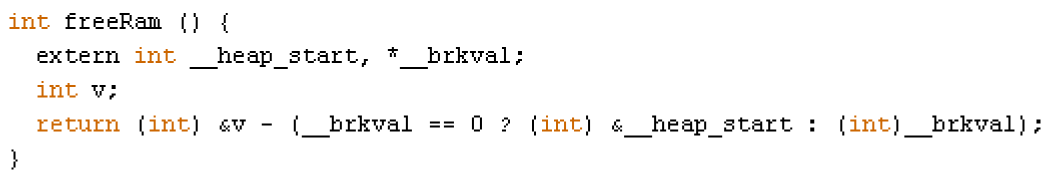
3) SRAM

SRAM usage is more dynamic and therefore more difficult to measure. The free_ram() function is one way to do this. By adding this function definition to the code, then call it from various places in the code the amount of free SRAM can be found.



```
void setup ()
{
  Serial.begin (57600) ;
  Serial.print ("testng mode") ;

  prekey_test () ;
  Serial.print ("Memory free: ") ; Serial.println(freeRam());
```

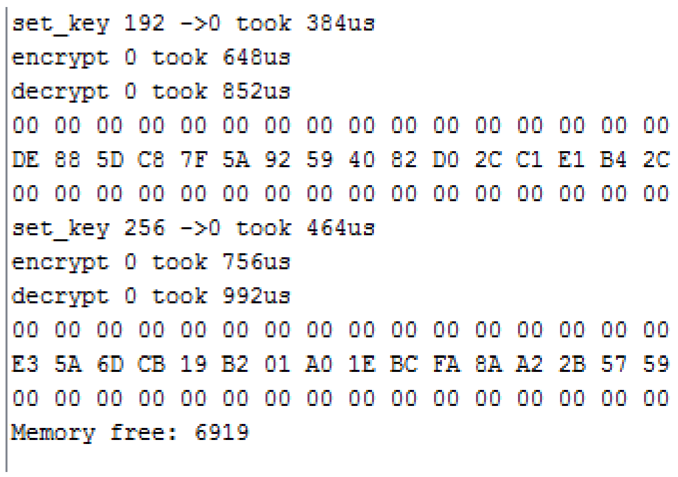
Fig 17: Calling free_ram( ) function in encryption /decryption code



```
int freeRam () {
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

Fig 18: free ram_ram( ) fuction in encryption/decryption code

For The function free_ram() actually reports the space between the heap and the stack but it does not report any de-allocated memory that is buried in the heap. Buried heap space is not usable by the stack, and may be fragmented enough that it is not usable for many heap allocations either. The space between the heap and the stack is what we really need to monitor if we are trying to avoid stack crashes.



```
set_key 192 ->0 took 384us
encrypt 0 took 648us
decrypt 0 took 852us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
DE 88 5D C8 7F 5A 92 59 40 82 D0 2C C1 E1 B4 2C
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
set_key 256 ->0 took 464us
encrypt 0 took 756us
decrypt 0 took 992us
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
E3 5A 6D CB 19 B2 01 A0 1E BC FA 8A A2 2B 57 59
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Memory free: 6919
```

Fig 19: output showing free SRAM memory

3.2.5 Summary of space available and space used:

Type of Memories	**Total space available**	**Space used**	**Space Remaining**
Flash	256 KB	8.79 KB	247.2 KB
EEPROM	4 KB	0.015 KB	3.98 KB
SRAM	8 KB	1.4 KB	6.9 KB

Table 3: Amount of flash, EEPROM and SRAM available/used/remaining

Chapter 4. Achievements of the project

Memory usage of different AES finalist algorithms has been calculated and the comparison of the memory usage between these algorithms is showed .AES was successfully installed in Arduino and was tested against many of the test-vectors (key varying, plaintext varying, Monte Carlo). It was able to perform encryption and decryption with varying key sizes!
This project was successful in attaining main goal i.e. to determine

- ✓ Whether AES can be installed in the Arduino platform: A simple AES encryption decryption process was installed.
- ✓ The amount of memory available for other device functionalities: Using separate techniques for observing the amount of memory used, we were able to find out the exact amount of memory that will be used/saved.

In addition to performing encryption and decryption, this project made sure that the time taken for the encryption/decryption is considerably low!! Time that encryption took was calculated at each and every step to make sure that we don’t face timing issues in future. Some speed/timing information is:

128 bit, key setup 0.37ms

128 bit, ECB, encryption 0.58ms / block (27.5kB/s)

128 bit, ECB, decryption 0.77ms / block (20.5kB/s)

192 bit, key setup 0.41ms

192 bit, ECB, encryption 0.71ms / block (22.5kB/s)

192 bit, ECB, decryption 0.92ms / block (17.5kB/s)

256 bit, key setup 0.52ms

256 bit, ECB, encryption 0.82ms / block (19.5kB/s)

256 bit, ECB, decryption 1.09ms / block (14.5kB/s)

Summary of timing information of encryption and decryption

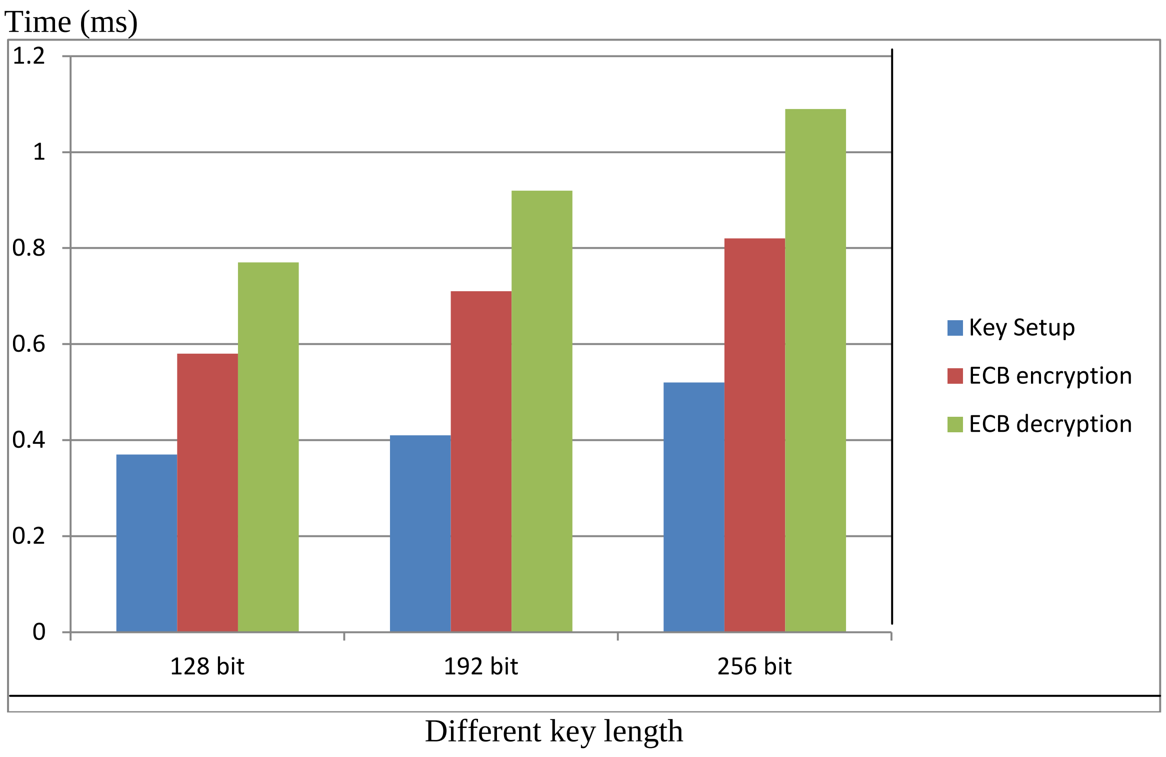


Fig 20: Timing information of encryption and decryption

From the above table we can see that the larger key size length the more time it took to key setup ,encrypt and decrypt data.

Chapter 5: Limitations

As the Arduino doesn't have a file structure and therefore there is no direct way to read/write to files. This implementation of AES in Arduino is done with the built in plain text provided by the AES library. To encrypt and decrypt of much larger plain text files using Arduino, extra Arduino SD card shield is needed with the SD card library. Data can easily be stored on an SD card and then using SD card library encryption and decryption can be done.

Chapter 6: Conclusion and Scope of Future Work

In conclusion, I want to say that, this project determined that AES can be implemented in Arduino and there will be sufficient space for other functionalities too.AES implementation took very less space and most of the space available in Arduino is unused and can be used for other functionalities. While dealing with time issues was not a part of this project, this project made sure that the time is kept in close watch. It was observed that the amount of time taken by encryption/decryption process is less and so ensures that implementing security wouldn't affect the efficiency of the device time wise.

My future work will be exploring further improvement of this project like using Arduino SD card shield and SD card library to test the encryption and decryption of text file using AES algorithm also I will try to use some other encryption modes like CTR. I have also a plan to implement AES in Raspberry Pi to get a comparison result between Arduino and Raspberry pi encryption and decryption performance.

References:

[1] V. Tharun Deep, Dr. Venkata Siva Reddy, "Comparative Analysis of AES Finalist Algorithms And Low Power Methodology For RC6 Block Cipher-A survey",

[2] Edward Chu, Paul Kim, Frank Liu, Jason Sharma, Jeffrey Yu," The Selection of the Advanced Encryption Standard "

[3] Mansoor Ebrahim, Shujaat Khan, Umer Bin Khalid, "Symmetric Algorithm Survey: A Comparative Analysis",

[4] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher", First Advanced Encryption Standard Conference.

[5] V. Tharun Deep, Dr. Venkata Siva Reddy, "Comparative Analysis of AES Finalist Algorithms And Low Power Methodology For RC6 Block Cipher-A survey", in International Journal for Technological Research In Engineering.

[6] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6TM Block Cipher", First AES Conference.

[7] Arduino Official Website, http://arduino.cc/

[8] The Laws Of Cryptography With Java Code by Neal R.Wagner.

[9] Ross Anderson, Eli Biham, Lars Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard", Cambridge University, England.

[10] KazumaroAoki and Helger Lipmaa, "Fast Implementations of AES Candidates" K'' uberneetika AS Akadeemia tee 21, 12618 Tallinn, Estonia.

[11] Bruce Schnier and Doug Whiting, "Performance Comparision of five AES finalists".

[12] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard", First Advanced Encryption

Standard Conference.

[13] Network Security, "Cryptography" , University of Houston, USA.

[14] J. Daemen and V. Rijmen, "AES Proposal: Rijndael", First Advanced Encryption Standard Conference.

[15] Najib A. Kofahil, Turki Al-Somani2 and Khalid Ai-Zamil3, Performance Evalution of Three Encryption/Decryption Algorithms.

[16] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., and Ferguson, N. , "Performance comparison of the AES submission."