CSE3009

Internet of Things (IoT)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Submitted to: Prof. Dheeba J

Project Title: Smart Home Management Systems and Security

Team Members:

Yash Priyadarshi	18BCE0442
Siddharth	18BCE0446
Reece D'Souza	18BCE0483
Navya Saxena	18BCE0770
Ronan D'Souza	18BCE0782

Table of Contents

Chapter 1

Introduction

Internet of things (IoT) is a kind of advanced information technology which has drawn societies' attention. This project presents different technologies involved in smart home management systems. After defining what smart home management is, it also explores different techniques and what are the current trends in this system. A comparative study is made on the security levels of each of the technologies. The different kinds of advantages and disadvantages, challenges faced in each kind of technique are also mentioned.

Smart-home devices hold a treasure trove of personal information, from your birth date to credit card details, that cybercriminals can steal via hacking if the devices lack robust protections to thwart attacks.

They can then use the stolen data to launch targeted attacks or sell the information to other cybercriminals. To mitigate this, the use of secure IoT Devices is necessary. To secure IoT devices, the use of encryption algorithms is necessary in the means of communication. Based on the research articles referred to, it was found that current security measures are useful, but consume either a lot of power, energy or have throughput constraints. The implementation of lightweight cryptography algorithms to a smart home IoT network is proposed. The proposed algorithm will be a hybrid algorithm based on the study of DES, 3DES and AES algorithm.

The broad area of this project is “Cybersecurity and the internet of things: vulnerabilities, threats, intruders and attacks”. The project expands on this idea by using encryption algorithms to secure IoT Devices.

The use of Internet of Things (IoT) devices has become very common these days, being present in almost every home. Internet of Things (IoT), to define it, is a huge network of interconnected gadgets which can gather data of the surroundings in which they operate. A simple example of an IoT device is a smoke detector, a device that is present in many homes. The term IoT was first coined by the British Technology Pioneer, Kevin Ashton in the year 1999. He had coined this term while he was working in an Auto-ID Lab Centre, and was referring to a global network of RFID or Radio-Frequency Identification connected objects.

Two main reasons that support the use of Lightweight Cryptographic Algorithms are mentioned as follows:

1. The efficiency of end-to-end communication

   Conventional Cryptographic algorithms cannot be applied to limited powered or battery-powered devices as their power consumption rate is high. Therefore, the use of lightweight cryptographic algorithms are necessary.

2. Applicability to lower resource devices

   The footprint of conventional cryptographic algorithms is far more than the lightweight algorithms and thus would open up various possibilities of connections with lower resource devices.

With this ideology, the implementation of lightweight cryptographic algorithms can be deemed fruitful in terms of power consumption, CPU load, execution time, etc.

1.1 Aim

The main aim of the project is as follows:

1. Measure the performance of these algorithms in terms of Execution Time and CPU Load.
2. Develop a new lightweight hybrid algorithm.
3. Measure its performance in terms of Execution Time and CPU Load.
4. Compare the performance of the conventional AES, DES, 3DES with the hybrid algorithm.

1.2 Objective

When it comes to Smart Home Security Systems, security measures are taken care of but security protocols are not followed in the communication system of the network. Smart home Security Systems have a large network with a lot of sensors continuously sending data throughout the network.

Power consumption in such a network is a major issue. The scope of the project is to perform a study on existing conventional cryptographic algorithms and their effect on IoT Devices. It also includes the creation of a hybrid algorithm that is lightweight, thus decreasing execution time and CPU load.

The input to these algorithms will be the same input plain text and secret key. The time will be measured using the Arduino IDE's inbuilt function `micros()`. The CPU load will be measured in the simulation software Proteus Professional. The power consumed cannot be measured as it is run in simulation software and the software will automatically adjust any loss of power. As the actual hardware components could be tested on, the algorithms will be run on the simulation software 'Proteus Professional', due to this any other performance metrics cannot be utilized.

1.3 Benefits

The system proposed an Encryption algorithm for Smart Home Security Systems using a Hybrid of SPECK, PRESENT and AES algorithms. After comparison with existing popular encryption algorithms in IoT - DES, 3DES and AES, the following benefits can be achieved:

- CPU Load – Lightweight
- Time Complexity - Less Execution Time
- Encryption - High Level of Security
- Resources - Particularly suited for resource constrained devices
- Incorporation - Easily incorporated in Smart Home System or any IoT Network

These benefits show that our proposed Hybrid algorithm should be efficient and flexible.

Chapter 2

Literature Survey

- M. Li, W. Gu, W. Chen, Y. He, Y. Wu and Y. Zhang, "Smart Home: Architecture, Technologies and Systems", Procedia Computer Science 131, 2018 [1]

Abstract – This paper examines the characteristics of smart homes, as well as the smart home's composition and application of key appliances, and also the smart home's key technologies, in order to demonstrate the design of a smart home's electricity service system and related communication systems.

Shortcomings – External network, gateway, and internal network are the three types of smart home communication systems. Security for the Smart Home has been accounted for but there is a chance that someone might attack through the external network and it is important that it is considered as well when implementing such an architecture.

- M. El-hajj, M. Chamoun, A. Fadlallah and A. Serhrouchni, "Analysis of Cryptographic Algorithms on IoT Hardware platforms", 2018 2nd Cyber Security in Networking Conference (CSNet) [2]

Abstract – Traditional cryptographic schemes might not be suitable to be implemented on resource limited IoT devices. This paper presents a benchmark of the most known cryptographic algorithms on the Raspberry Pi platform, with a comparison with Arduino benchmark results provided in the literature.

Shortcomings – Connectivity challenges for billions of devices to intercommunicate, the number of connected objects and their wide range of deployment and applications. Power consumption is also another problem.

- A. Safi, "Improving the Security of Internet of Things Using Encryption Algorithms", International Journal of Computer & Information Engineering, Vol. 11, No. 5, 2017 [3]

Abstract – In this paper, a hybrid encryption algorithm which has been conducted in order to reduce safety risks and enhance encryption speed and less computational complexity has been introduced. The purpose of this hybrid algorithm is information integrity, confidentiality, nonrepudiation in data exchange for IOT.

Shortcomings – The hybrid algorithm uses NTRU as one part of the algorithm, which requires longer keys and cryptograms.

- M. Dragos, Y. Chen, P. and P. Musilek, "IoT-based smart homes: A review of system architecture, software, communications, privacy and security", Internet of Things 1 (2018) [4]

Abstract – This paper focuses on smart home management system software solutions and modules, related communication technologies, and privacy and security concerns associated with the connected design of modern smart homes.

Shortcomings – The development of lightweight algorithms for local data processing and reducing the number of transmissions between devices is required to optimise communication among Smart Home devices.

- V. A. Thakor, M. A. Razzaque and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource- Constrained IoT Devices: A Review, Comparison and Research Opportunities", IEEE Access, IEEE, 2021 [5]

Abstract – This paper focuses on resource constraint IoT devices (RFID tags, sensors, smart cards, etc.). Lightweight cryptography algorithms have been implemented for security purposes which also focuses on optimizing the energy consumption and smooth communication.

Shortcomings – The use of just one S-Box for the AES algorithm can make the algorithm less computationally complex.

- S. Surendran, A. Nassef, and B. D. Beheshti, "A Survey of Cryptographic Algorithms for IoT Devices", 2018 IEEE Long Island Systems, Applications, and Technology Conference (LISAT) [6]

Abstract – This article explains how lightweight cryptography is meant for extremely resource-constrained devices and they are used not only for encryption purposes but also for authentication and hashing under highly resource-constrained devices. The article starts by explaining various cryptography algorithms after which the authors explain the attacks on lightweight ciphers. Lastly, a performance comparison in terms of execution time is shown for the algorithms chosen.

Shortcomings – This article has not proposed any algorithm, only a comparative study between different algorithms has been analysed and shown as the results.

- J. S. Kumar and D. R. Patel, "A Survey on Internet of Things: Security and Privacy Issues", IEEE Internet of Things Journal [7]

Abstract – With the increase in the number of IoT devices, there has been a tremendous increase in the amount of data to be processed and analysed. When such large data are processed there is a high risk of threats and security issues to the IoT devices in use. This

article reviews the threats, challenges, attack vectors of IoT networks and their security requirements. A combination of network-based IoT architecture and Software Defined Networking has been proposed. Finally, to provide security solutions a detailed overview of Software-Defined Security has been explained.

Shortcomings – The framework proposed by the authors for countermeasure is very big and computationally extensive due to which the implementation cannot be carried out on a large number of devices having less performance power.

- W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-Defined Security", IEEE Internet of Things Journal [8]

Abstract – With the increase in the number of IoT devices, there has been a tremendous increase in the amount of data to be processed and analysed. When such large data are processed there is a high risk of threats and security issues to the IoT devices in use. This article reviews the threats, challenges, attack vectors of IoT networks and their security requirements. A combination of network-based IoT architecture and Software Defined Networking has been proposed. Finally, to provide security solutions a detailed overview of Software-Defined Security has been explained.

Shortcomings – The framework proposed by the authors for countermeasure is very big and computationally extensive due to which the implementation cannot be carried out on a large number of devices having less performance power.

- J. Bugeja, A. Jacobson and P. Davidson, "On Privacy and Security Challenges in Smart Connected Homes", 2016 European Intelligence and Security Informatics Conference [9]

Abstract – This article has presented an overview of security and privacy challenges on IoT networks. The difference being that article 10 was focused on the broader area of IoT whereas this article is focused on the smart home domain. The constraints have been identified, solutions have been evaluated and challenges have been discussed in this article for the privacy and security issues.

Shortcomings – The article did not provide an implementation model for the proposed solution based on the survey conducted.

- M. Abomhara and G. M. Køien, "Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks", Journal of Cyber Security and Mobility [10]

Abstract – This article classifies threat types, analyzes them, and characterizes the intruders and attacks on IoT devices and their services. The ACID properties have also been

discussed after which various security threats, attacks, and vulnerabilities of IoT devices have been raised. Lastly, the authors have discussed intruders and organized groups for attacks.

Shortcomings – The article is a survey on various threats and vulnerabilities on IoT networks. It was focused on security challenges on IoT devices and services and failed to provide a real time solution for the classified attacks.

Chapter 3

Components Used

3.1 Sensors and Actuators

1. Universal Keyboard

It is a set of buttons arranged in a block or pad which bear digits, symbols or alphabetical letters. Pads mostly containing numbers and used with computers are numeric keypads.

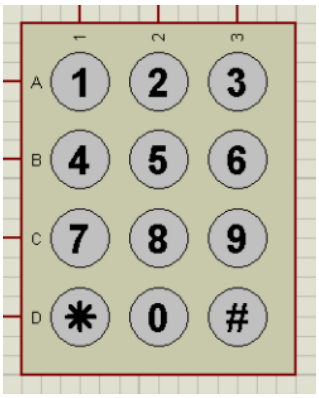

Fig. 1: Universal Keyboard

Keypads are found on devices which require mainly numeric input such as calculators, television remotes, push-button telephones, vending machines, ATMs, Point of Sale devices, combination locks, and digital door locks.

2. Servo Motor

A servo motor is essentially a coupling of a DC motor to a potentiometer such that the control circuit is able to establish the absolute position of the rotor.

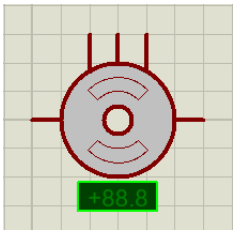

Fig. 2: Servo Motor

Typically, the shaft of the DC motor will be geared down to drive the cam arm so that a substantial amount of torque can be developed.

3. 16x2 LCD

LCD stands for Liquid crystal display, 16x2 LCD has 16 columns and 2 rows so it will have 16x2=32 characters in total and each character will be made of 5x8 pixel dots.

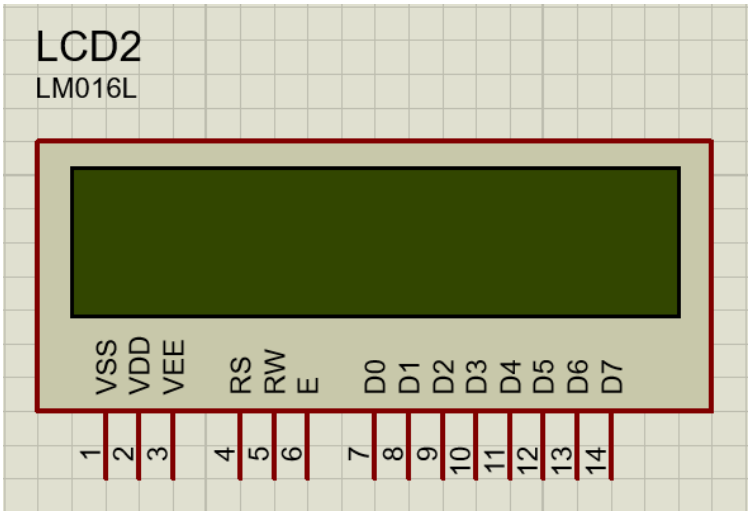

Fig. 3: 16x2 LCD

This is an Alphanumeric LCD display module, which means it can display alphabets and numbers.

4. Arduino UNO

Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button.

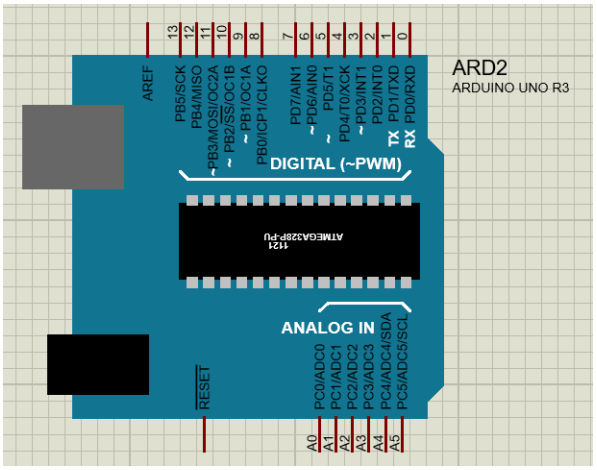

Fig. 4: Arduino UNO

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.

5. Virtual Terminal

Virtual Terminal is a very useful tool available in the Proteus. With the help of Virtual Terminal one can easily simulate the serial communication that he / she use in his / her embedded systems.

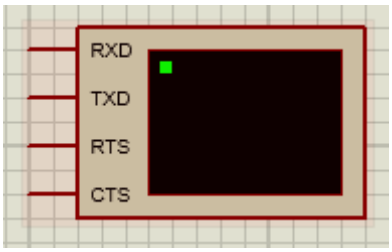

Fig. 5: Virtual Terminal

It is important to note here that almost every microcontroller that is used in the embedded system has integrated UART (Universal Asynchronous Receiver Transmitter) on it which is used to perform serial communication between other hardware used in embedded system that also supports Universal Asynchronous receiving and transmission. The Virtual Terminal in the Proteus is bi-directional which means that it can send and receive data simultaneously.

3.2 Working Model of all Sensors

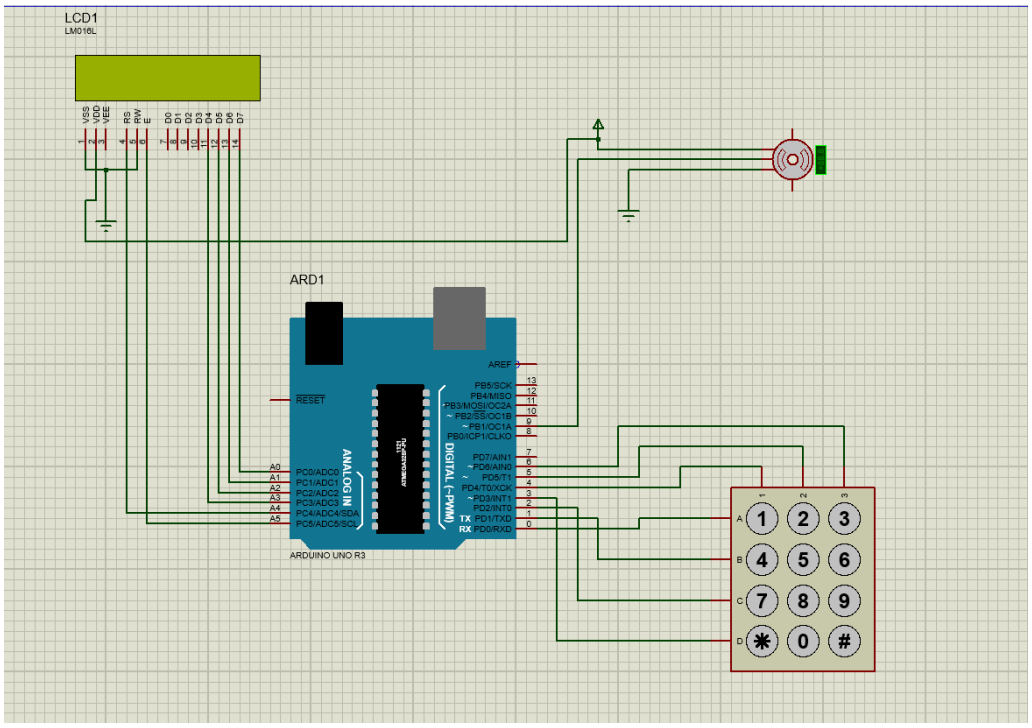

Fig. 6: Circuit Diagram of all Sensors and Actuators

- The *Universal Keypad* is used to enter the PIN Code for the door locking system.
- The *LCD Screen* is used to display any outputs given by the Arduino UNO.
- *Arduino UNO* is used to verify the PIN Code and make necessary changes to the *Servo Motor* and display accordingly on the *LCD Screen*.
- The *Servo Motor* is used to simulate the locking and unlocking of a door.

Chapter 4

Proposed Architecture

Novel Lightweight Hybrid Algorithm

The Hybrid Algorithm is made with the combination of all the lightweight aspects of each of the algorithms and more. Therefore, the novel algorithm is lightweight without compromising any aspect of security.

Key Generation:

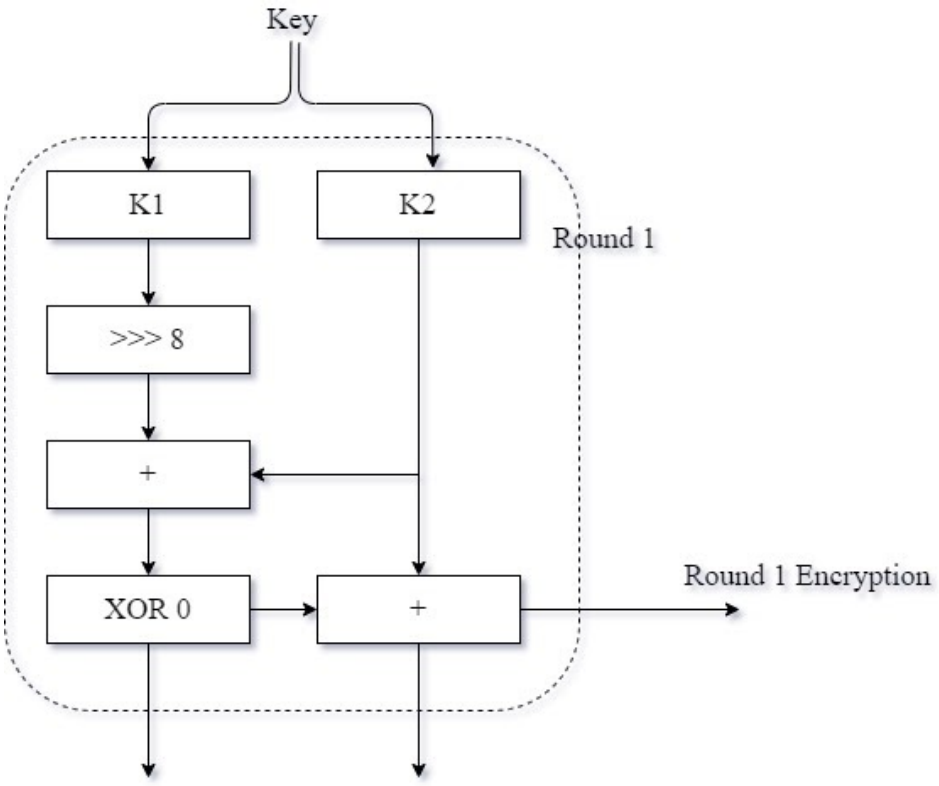

Fig. 7: Hybrid Algorithm Key Generation Function

1. The key is split into two K1 and K2
2. In each round (11 rounds in total):
    a. K1 undergoes an 8-bit right shift
    b. Addition with K2
    c. XOR'd with 0
    d. K2 undergoes addition with an output of c
    e. The output of d is the key that will be used in the Encryption Procedure

3. The output of K1 and K2 received from each round acts as the input for the next round.
4. Thus, the keys are generated.

Encryption Procedure:

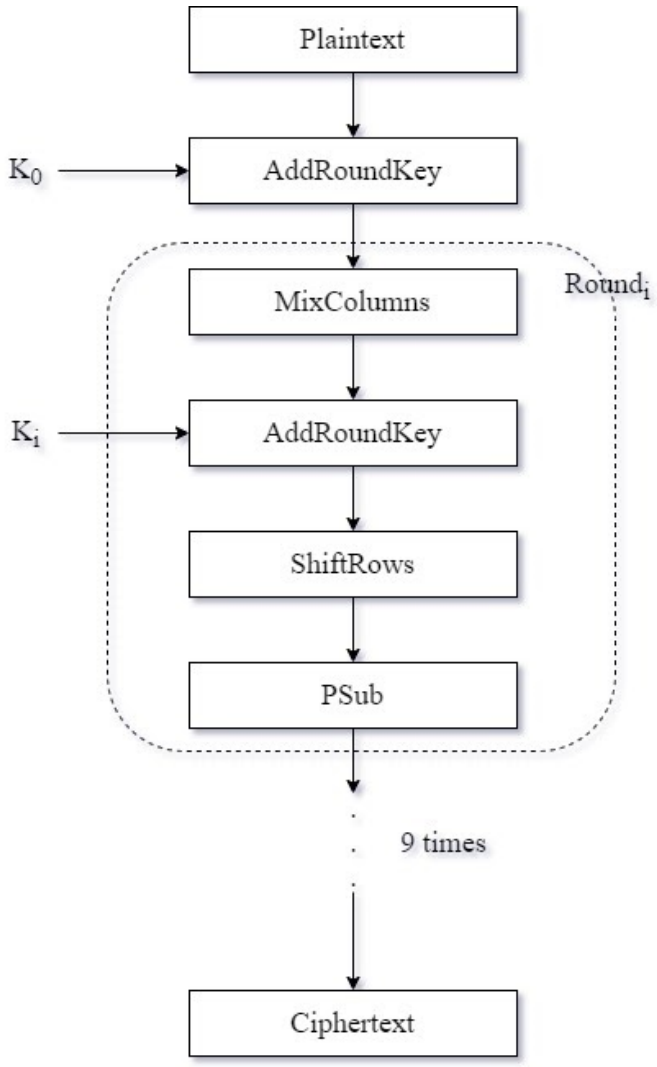

Fig. 8: Hybrid Algorithm Encryption Algorithm

1. The plaintext (PT) is fed into the algorithm. It is 64-bit in length
2. The AddRoundKey() is performed with the PT and Key0 from the key generation procedure and the output is termed as a ‘state’ variable.
3. In each round (for a total of 10 times):
    a. Mix Columns: This step belongs to the AES Algorithm. Here, similar to the AES Algorithm, each column of the 64 bits is transformed into new bytes using a special mathematical function.

- For every column:
- Assign the column into a temporary variable.
- Each column is multiplied by the following Matrix:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

b. AddRoundKey: This step belongs to the PRESENT Algorithm. Here, the 64-bit value of the 'state' and the 64-bit Key for that round are XOR'd.

c. Shift Rows: This step belongs to the AES Algorithm. Here, similar to the AES Algorithm, each state undergoes the following operations:

   i. The first row remains unchanged
   ii. The second row is shifted to the left by one position
   iii. The third row is shifted to the left by two positions
   iv. The fourth row is shifted to the right by one position

d. PSub: This step belongs to the PRESENT Algorithm. Here, similar to the PRESENT Algorithm, each element is permuted to position in the PBox values from the PRESENT Algorithm. The PBox values are presented below:

Table 1: Permutation Box of PRESENT Algorithm

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

4. After 10 rounds in total, the state variable is the ciphertext.

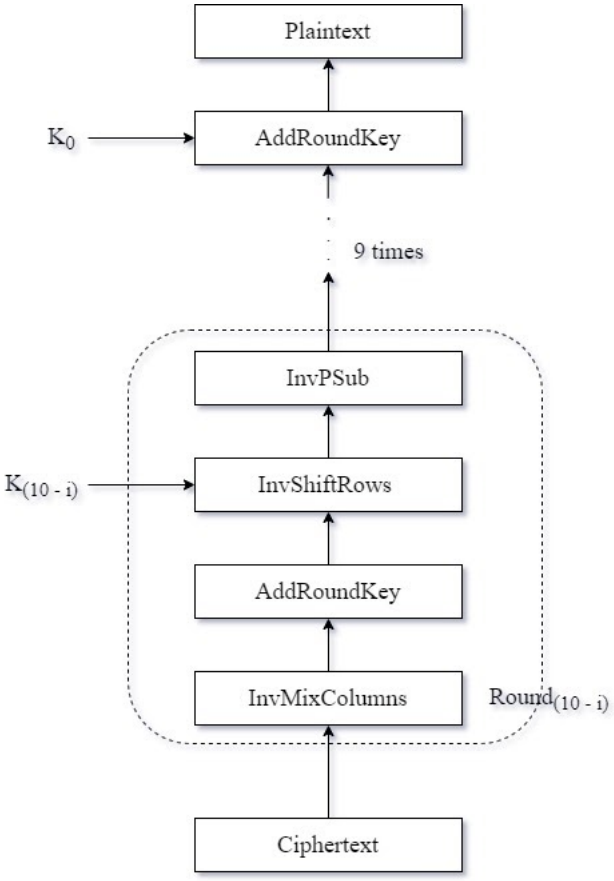

Fig. 9: Hybrid Algorithm Decryption Algorithm

5. For decryption, the order of each round is as follows:
    a. InvPSub: This step belongs to the PRESENT Algorithm. Here, similar to the PRESENT Algorithm, each element is permuted to position in the PBox values from the PRESENT Algorithm. The PBox values are presented below:

Table.2: Inverse Permutation Box of PRESENT Algorithm

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I(i)	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
I(i)	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
I(i)	2	6	10	14	18	22	25	30	34	38	42	46	50	54	58	62
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
I(i)	3	7	11	15	19	23	26	31	35	39	43	47	51	55	49	63

b. Inv Shift Rows: This step belongs to the AES Algorithm. Here, similar to the AES Algorithm, each state undergoes the following operations:

   - The first row remains unchanged
   - The second row is shifted to the right by one position
   - The third row is shifted to the right by two positions
   - The fourth row is shifted to the left by one position

c. AddRoundKey: This step belongs to the PRESENT Algorithm. Here, the 64-bit value of the 'state' and the 64-bit Key for the round ($10 - Round_i$) are XOR'd.

d. Inv Mix Columns: This step belongs to the AES Algorithm. Here, similar to the AES Algorithm, each column of the 64 bits is transformed into new bytes using a special mathematical function.

   - For every column:
   - Assign the column into a temporary variable.
   - Then each column is multiplied by a matrix:

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

6. Finally, the key0 is added using the AddRoundKey function.
7. The state variable obtained is the original plaintext.

Workflow for Implementation of Project

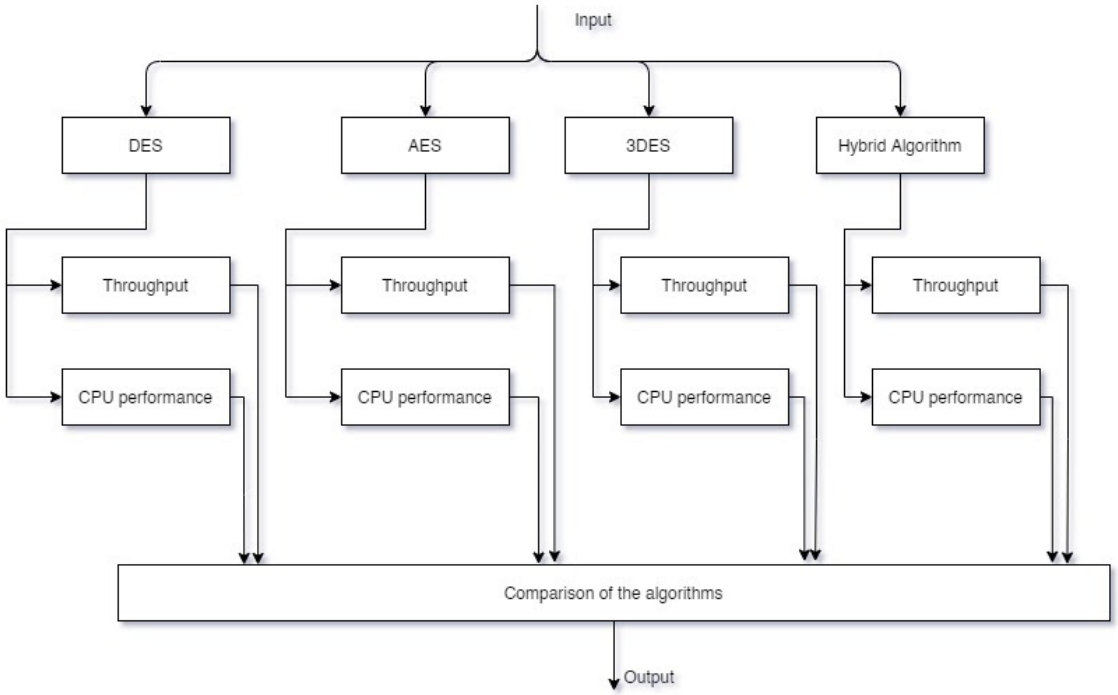

Fig. 10: Workflow of the project

Input: This is the input given to all the algorithms. The input possesses the same value for all the algorithms, including the novel hybrid algorithms. This can help accurately measure the performance metrics. More inputs can yield more outputs and therefore achieve higher accuracy.

DES: This refers to the DES Algorithm. DES stands for Data Encryption Standard. It is a symmetric block cipher algorithm with 64-bit plain text and 56-bit keys. It consists of 16 rounds of encryption and a similar number for decryption.

AES: This refers to the AES Algorithm. AES stands for Advanced Encryption Standard. It is based on the principle of 'substitution–permutation'.

3DES: This refers to the Triple-DES Algorithm. Triple DES is an advanced version of the DES algorithm. It applies the DES algorithm three times to each data block.

Hybrid Algorithm: This refers to the Hybrid Algorithm that is developed in the project. It has a 64-bit plaintext input and a 128-bit key.

Throughput: The throughput refers to the execution time taken by each algorithm. This is one of the performance metrics utilized for the comparison study.

CPU Load/Performance: The CPU Load refers to the measure of computational work that a computer system performs. The load average refers to the average value of the same measure over some time.

Comparison of the algorithms: The comparison of the algorithms is the study performed in the project. This comparison is done after obtaining the simulation results.

Output: The output is the verification of the algorithms as well as the results of the comparison study done in the previous step.

Chapter 5

Simulation Implementation

Cryptographic Algorithms Analysis

To implement the project, the following is performed:

1. Open Arduino IDE
2. Add the libraries into Arduino IDE (if necessary)

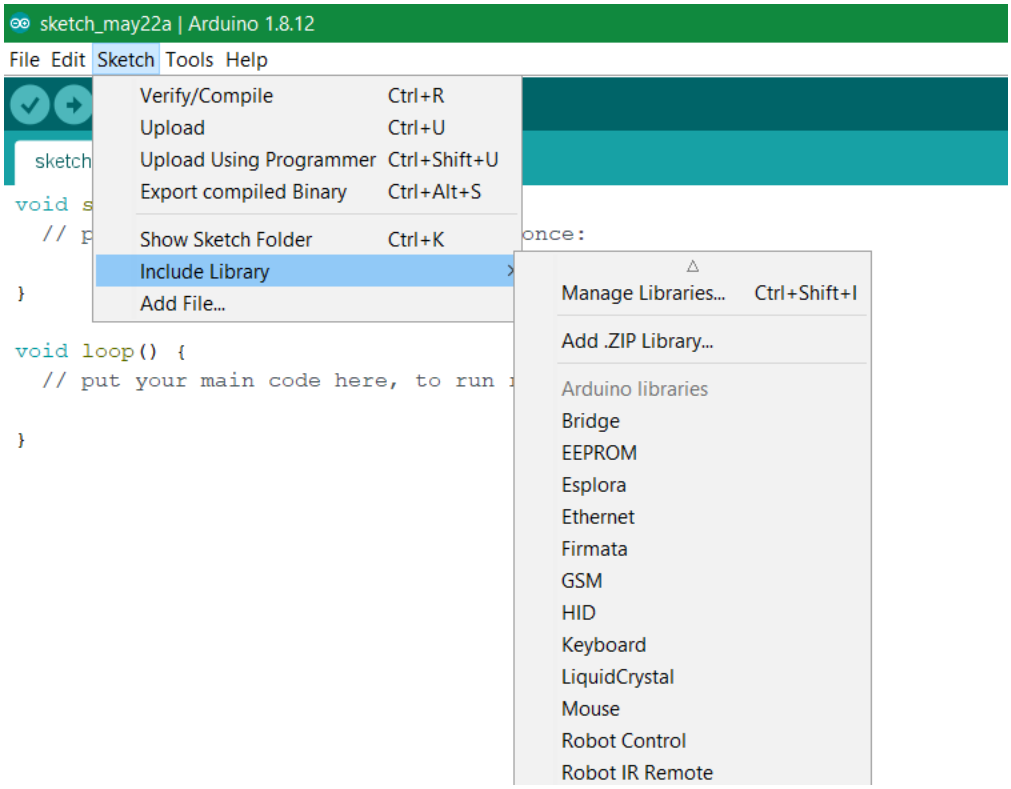

Fig. 11: Location of Include Library option in Arduino IDE

3. Add the Code for the specific algorithms:

DES:

Refer Appendix A

3DES:

Refer Appendix A

AES:

Refer Appendix B

Hybrid:

Refer Appendix C

4. Verify the code and generate the .HEX File.

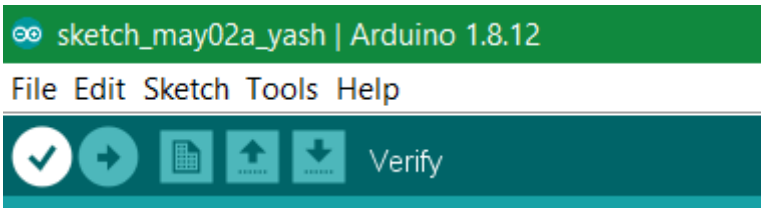

Fig. 12: Location of Verify-in Arduino IDE

5. To verify if the .HEX File is generated or not, check the terminal.

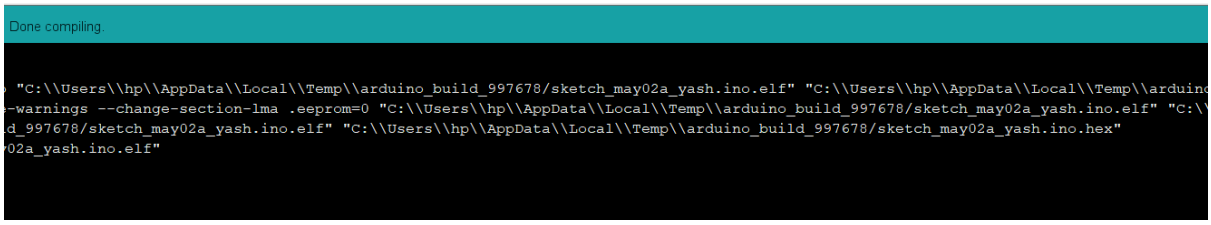

Fig. 13: Verification of .HEX file in Arduino IDE

6. Open Proteus Professional and create the following circuit:

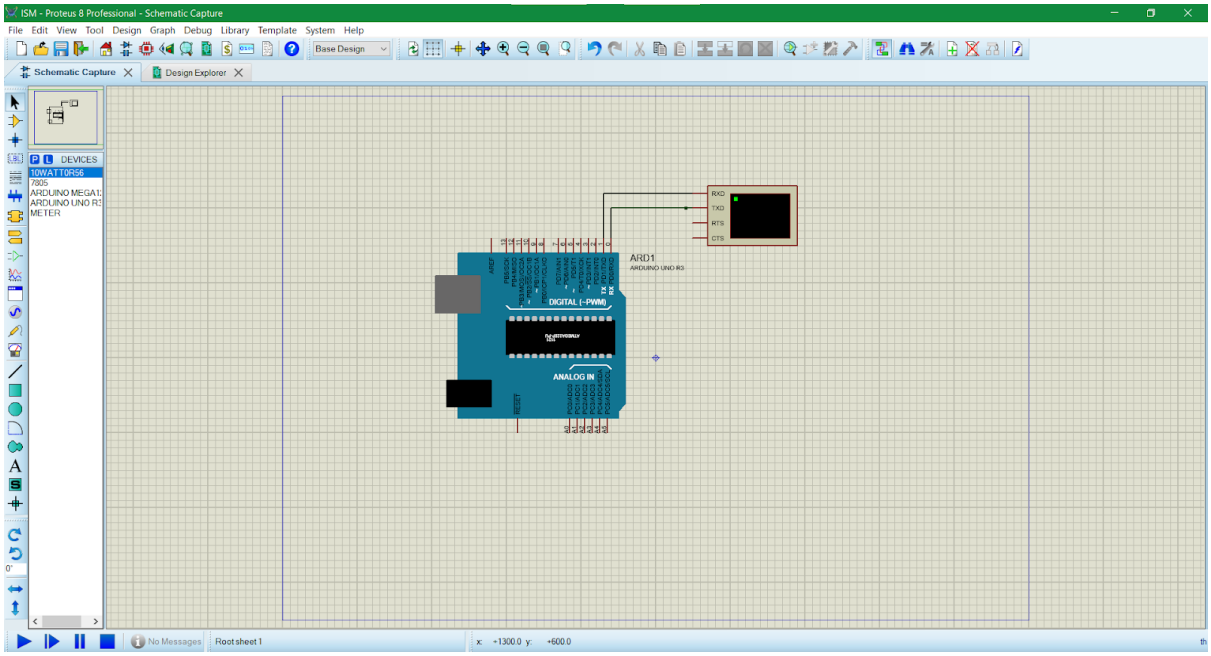

Fig. 14: Circuit to be created using Proteus Professional

7. Add the .HEX File to Arduino UNO Component.

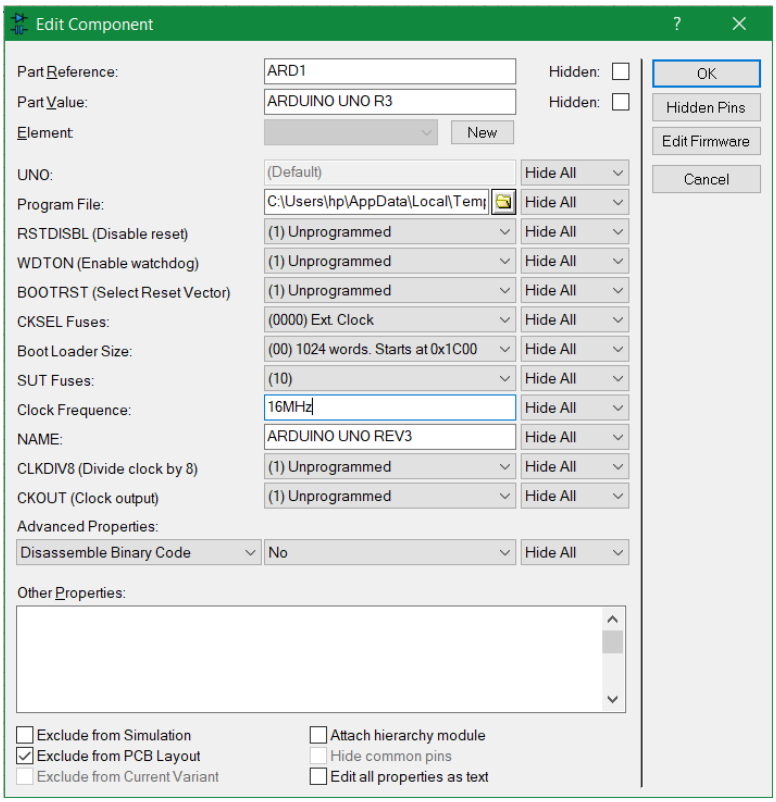

Fig. 15: Edit Component Window of Arduino UNO

8. Change the Program File path to the .HEX File's location.

[Note: The location can be found in the terminal of the Arduino IDE after verifying it.**]**

9. To simulate the circuit, press the Run ('play' symbol) button on the bottom left.

DES & 3DES:

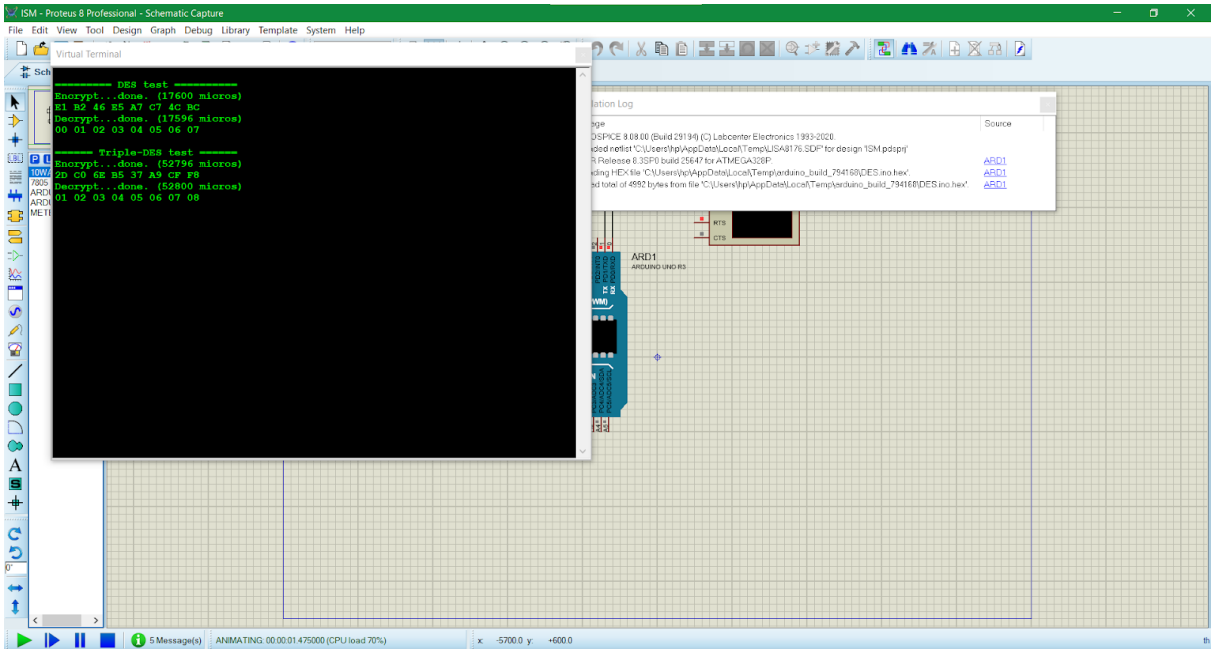

Fig. 16: Implementation of DES & 3DES Algorithm

AES:

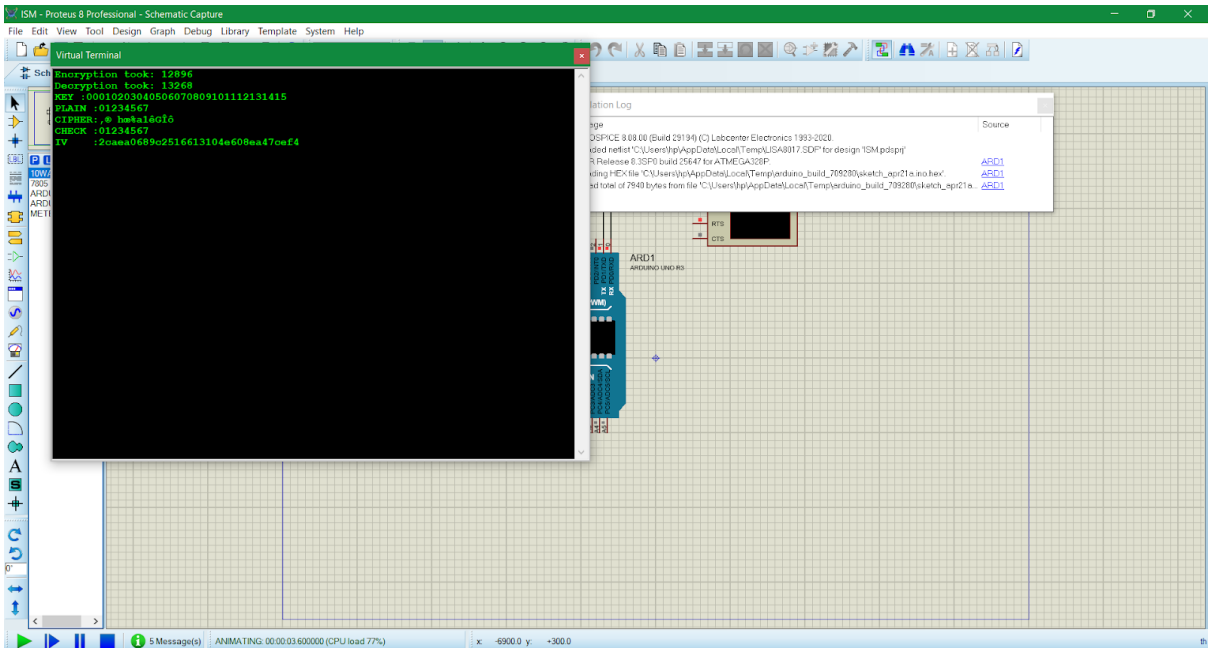

Fig. 17: Implementation of AES Algorithm

Hybrid Algorithm:

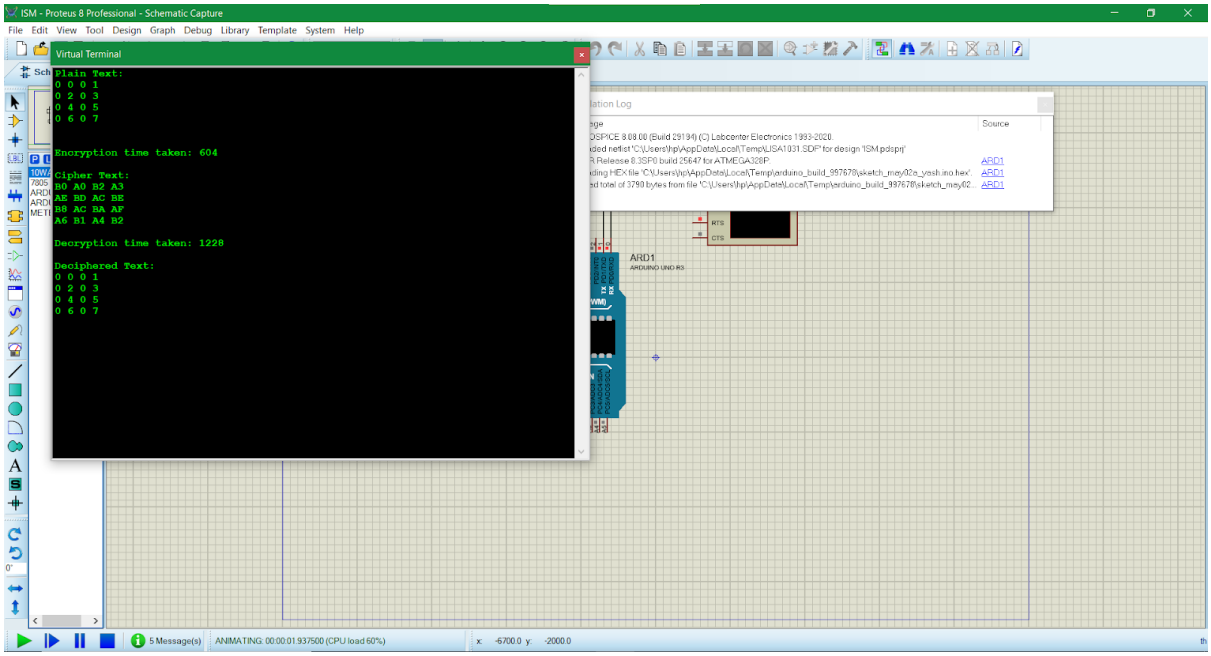

Fig. 18: Implementation of Hybrid Algorithm

Implementation of Novel Algorithm in Door Locking System

To run the project following steps are need to be followed:

- Open the proteus file containing the project components which are connected together.
- Verify the Arduino code and copy the hex file path from the console
- Paste the hex file path into the Arduino code
- Run the simulation
- In the keypad enter the 4-digit code and press ‘#’ to enter
- If the entered code is equal to password which is ‘0123’ then the motor attached to the Arduino will start rotating which indicates that the lock is opened.
- The LCD display will show ‘Access Granted’ but it will remain opened only for 5 seconds displaying ‘GET WITHIN’ time remaining & then it will close again with the message ‘RE-LOCKING’.
- If the password is incorrect then the lcd will display ‘INCORRECT’ followed by ‘Get Away’.

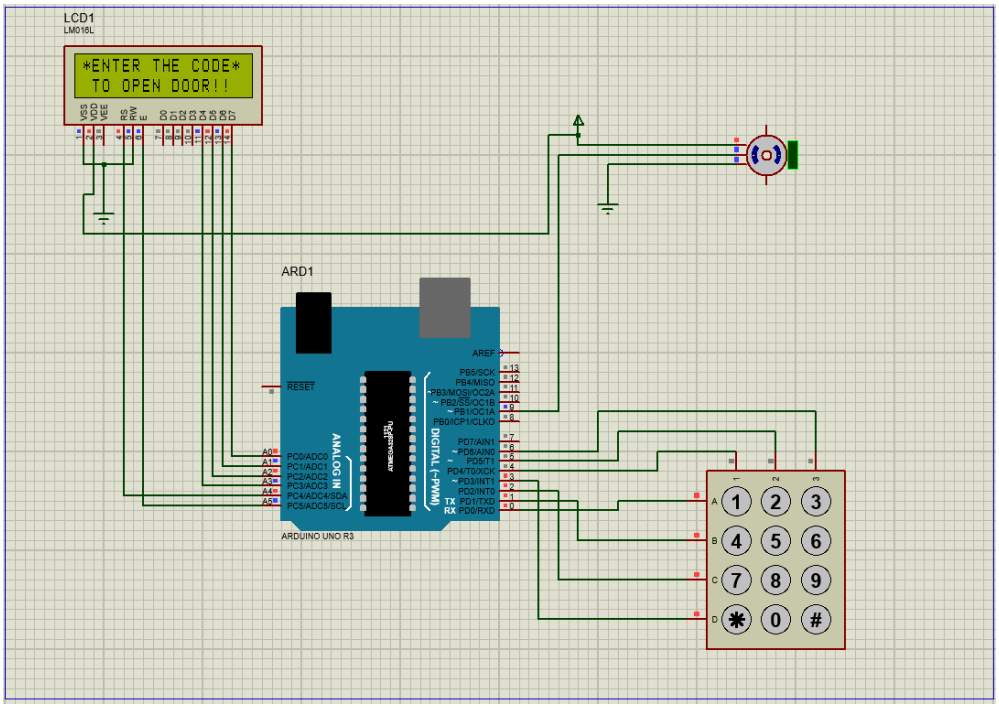

Fig. 19: Door Locking System – Start-up

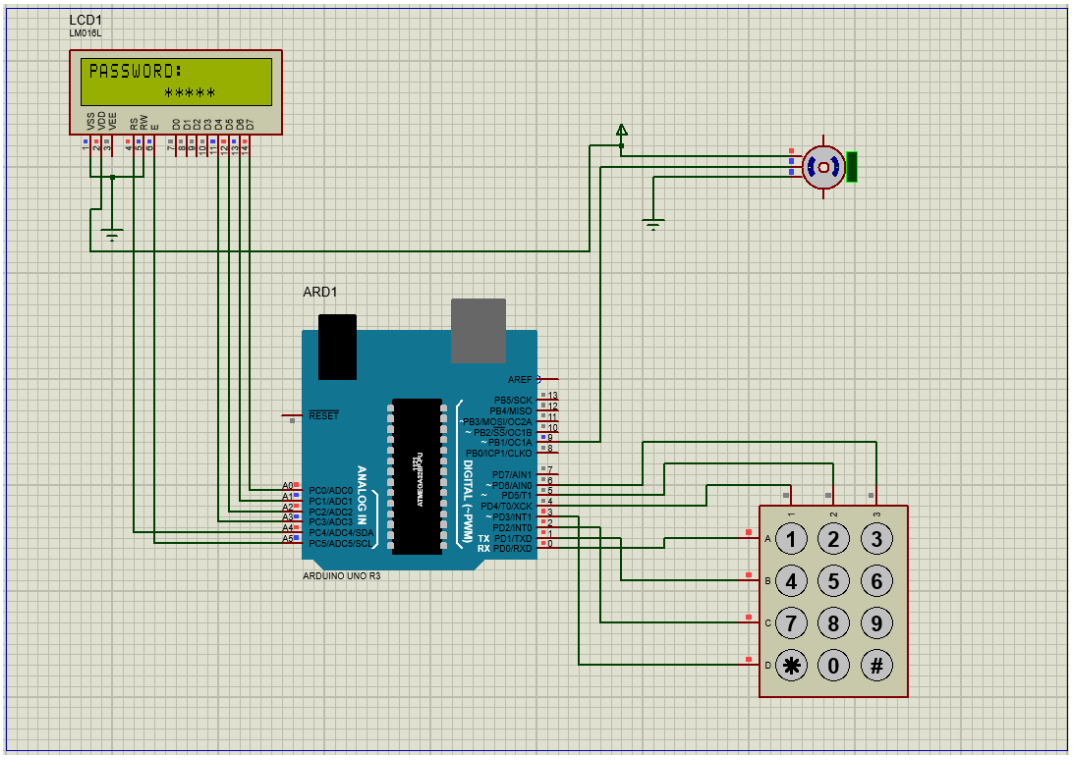

Fig. 20: Door Locking System – Entering Password

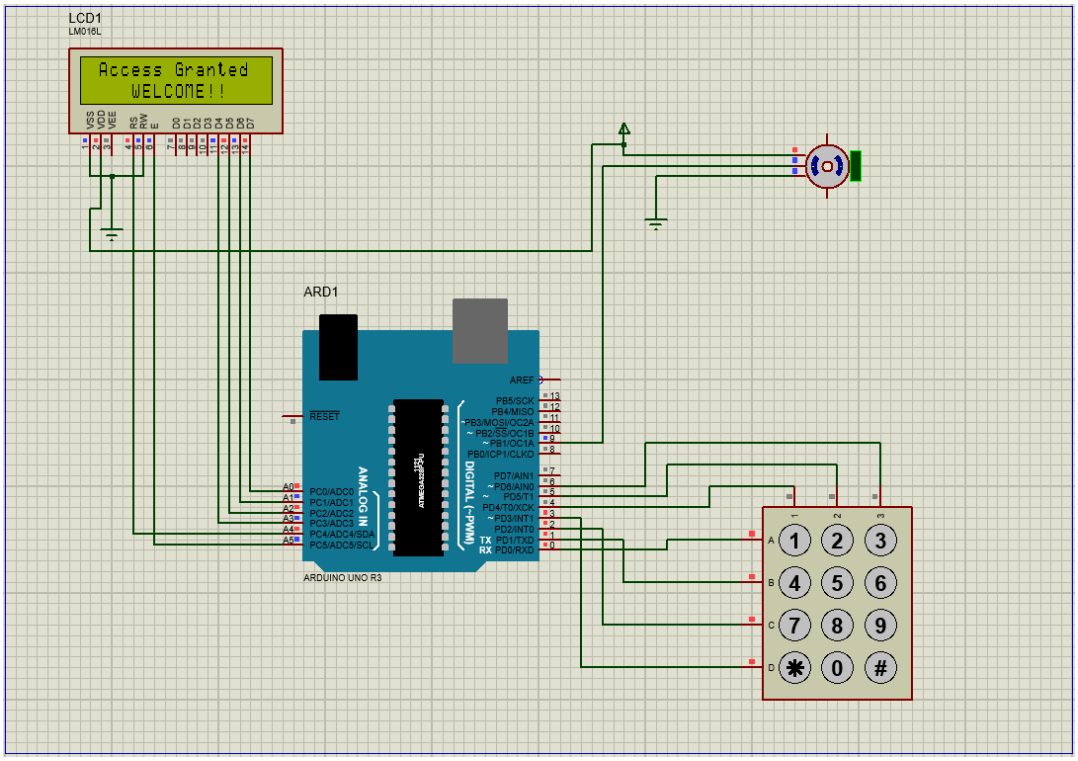

Fig. 21: Door Locking System – Password Correct

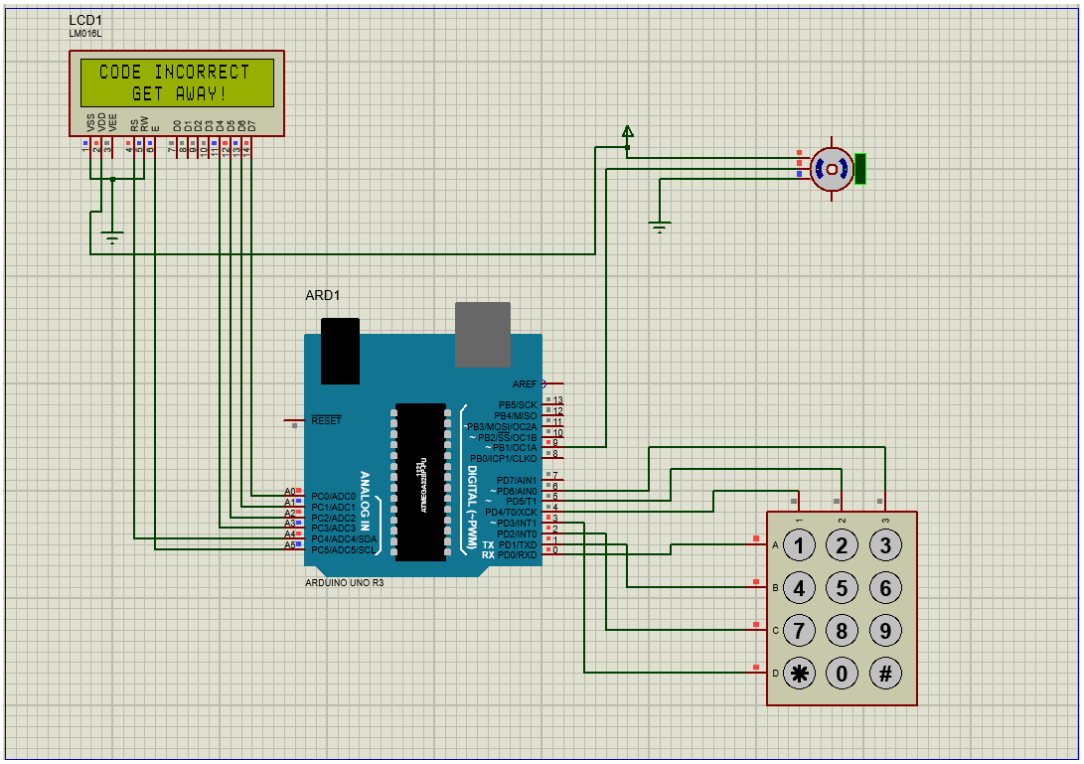

Fig. 22: Door Locking System – Password Incorrect

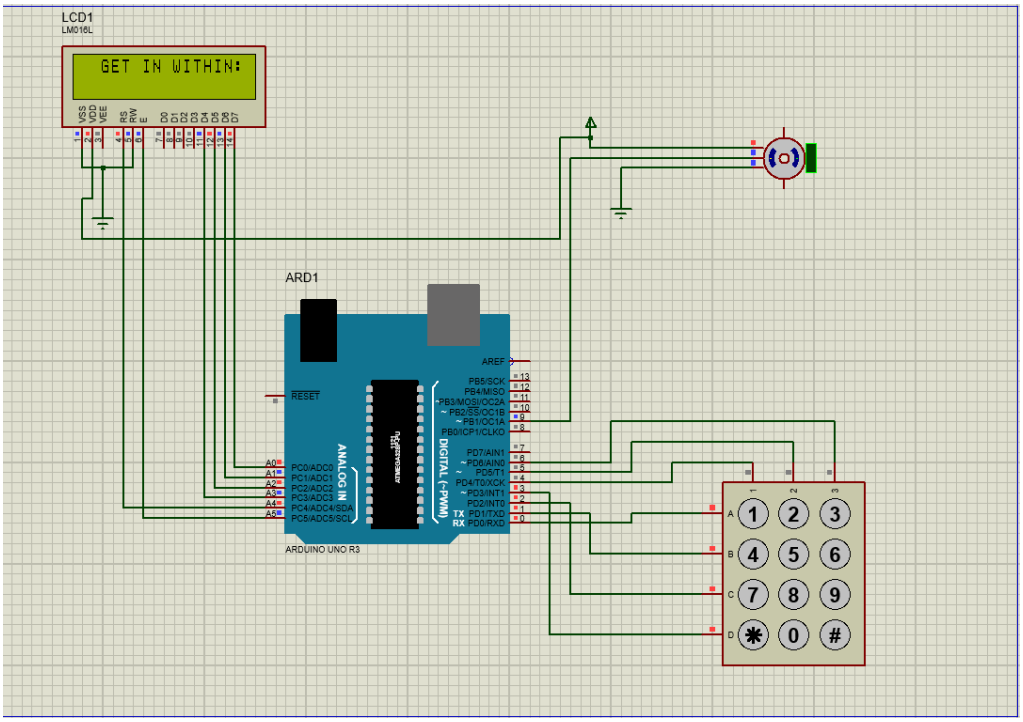

Fig. 23: Door Locking System – Relocking

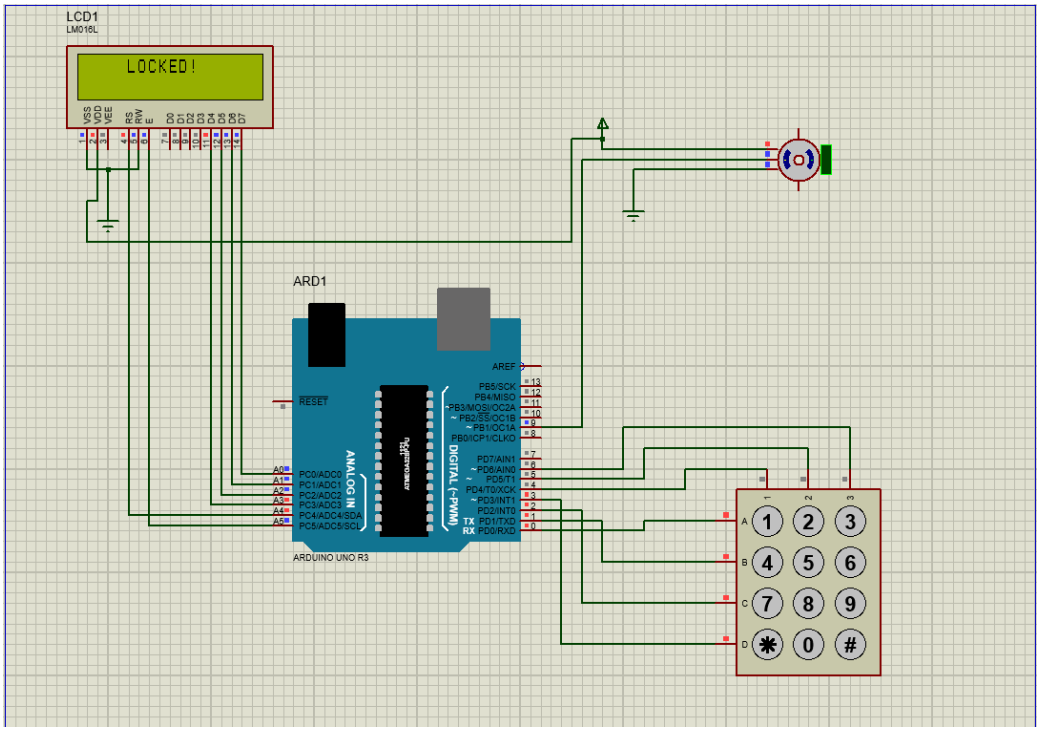

Fig. 24: Door Locking System – Locked

The code for the same is available in Appendix D.

Chapter 6

Results & Discussions

Encryption Execution Time of all the algorithms

Table. 3: Encryption Execution Time of all the algorithms

Algorithm	Execution Time (in µs)
DES	~17600
3DES	~52800
AES	~12800
Hybrid	~750

Graphical Representation of Encryption Execution Time of all the algorithms

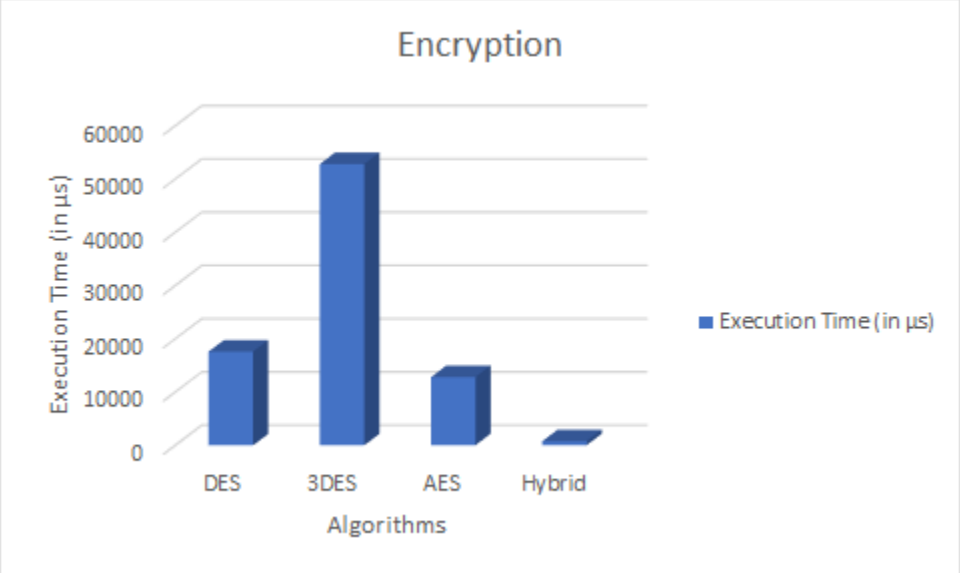

Fig. 25: Graphical Representation of Encryption Execution Time

Decryption Execution Time of all the algorithms

Table. 4: Decryption Execution Time of all the algorithms

Algorithm	Execution Time (in µs)
DES	~17600
3DES	~52800
AES	~13000
Hybrid	~1200

Graphical Representation of Decryption Execution Time of all the algorithms

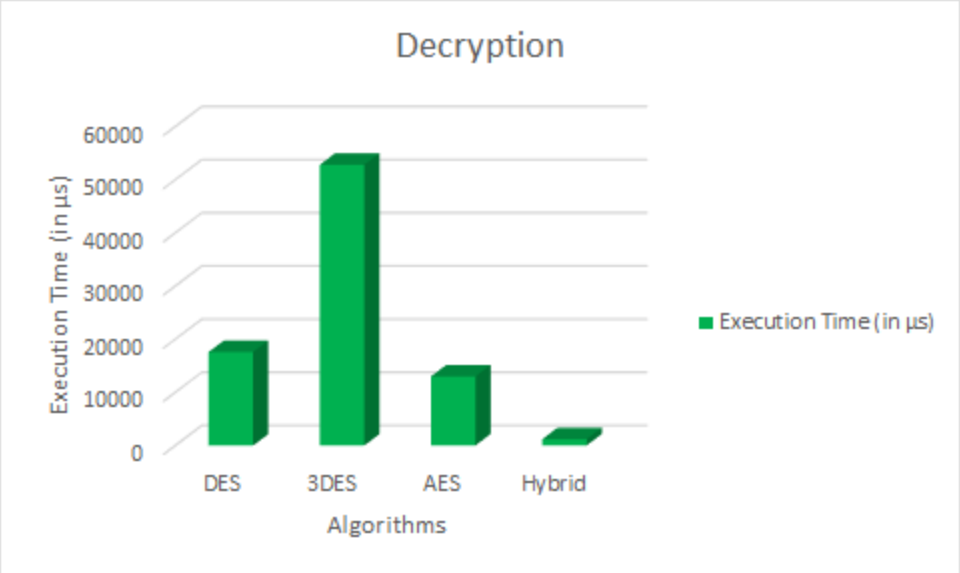

Fig. 26: Graphical Representation of Decryption Execution Time

CPU Load of all the algorithms

Table. 5: CPU Load of all the algorithms

Algorithm	CPU Load (%)
DES	~50

3DES	~51
AES	~45
Hybrid	~38

Graphical Representation of CPU Load

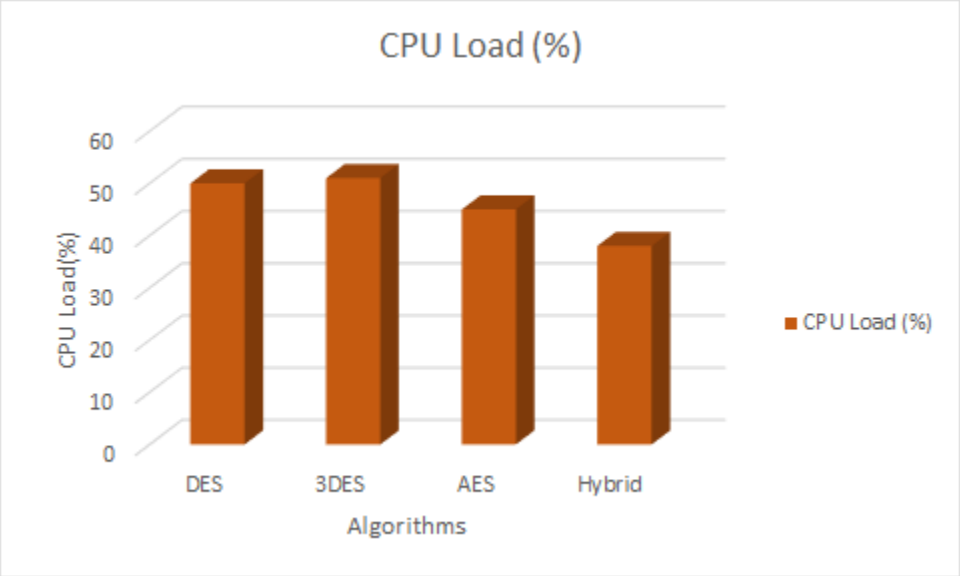

Fig. 27: Graphical Representation of CPU Load

Door Locking System

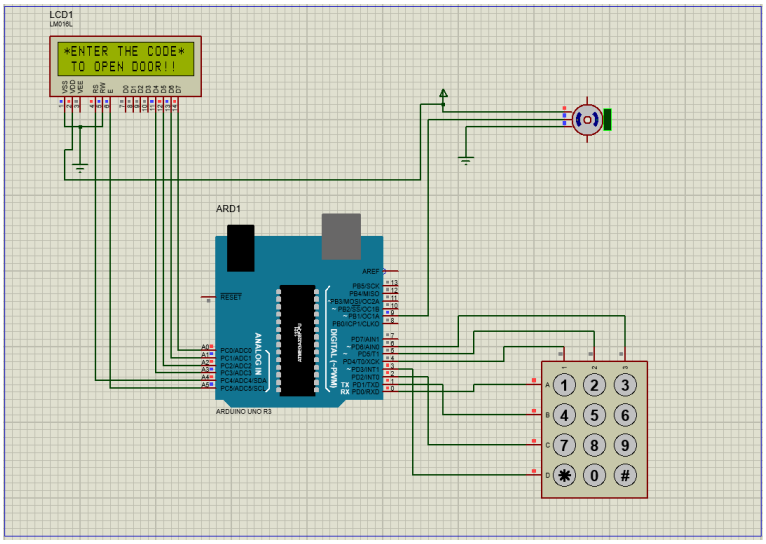

Fig. 28: Implementation of Door Locking System with Hybrid Algorithm

Chapter 7

Conclusion

As clearly visible from Table 3 and Table 4, the Hybrid Algorithm's encryption and decryption execution times are significantly lesser than the conventional algorithms. The CPU Load when the Hybrid Algorithm is being executed is also quite significantly lower than when the other conventional algorithms are executed.

This is due to the addition of several lightweight features in a normal heavyweight algorithm, such as reduction in the number of rounds, reduced size of substitution boxes, and optimization of C++ code.

The following setup was run in an Arduino Environment, but the results would hold the same for any other microprocessor environment (E.g. Raspberry Pi).

The system proposed an Encryption algorithm for Smart Home Security Systems using a Hybrid of SPECK, PRESENT and AES algorithms. After comparison with existing popular encryption algorithms in IoT - DES, 3DES and AES, the following benefits can be achieved:

- CPU Load – Lightweight
- Time Complexity - Less Execution Time
- Encryption - High Level of Security
- Resources - Particularly suited for resource constrained devices
- Incorporation - Easily incorporated in Smart Home System or any IoT Network

These benefits show that our proposed Hybrid algorithm should be efficient and flexible.

Thus, the expected result from the Problem Statement Section has been validated and thus the scope of the project achieved.

References

[1] M. Li, W. Gu, W. Chen, Y. He, Y. Wu and Y. Zhang, “Smart Home: Architecture, Technologies and Systems”, Procedia Computer Science 131, 2018

[2] M. El-hajj, M. Chamoun, A. Fadlallah and A. Serhrouchni, “Analysis of Cryptographic Algorithms on IoT Hardware platforms”, 2018 2nd Cyber Security in Networking Conference (CSNet)

[3] A. Safi, “Improving the Security of Internet of Things Using Encryption Algorithms”, International Journal of Computer & Information Engineering, Vol. 11, No. 5, 2017

[4] M. Dragos, Y. Chen, P. and P. Musilek, “IoT-based smart homes: A review of system architecture, software, communications, privacy and security”, Internet of Things 1 (2018)

[5] V. A. Thakor, M. A. Razzaque and M. R. A. Khandaker, “Lightweight Cryptography Algorithms for Resource- Constrained IoT Devices: A Review, Comparison and Research Opportunities”, IEEE Access, IEEE, 2021

[6] S. Surendran, A. Nassef, and B. D. Beheshti, “A Survey of Cryptographic Algorithms for IoT Devices”, 2018 IEEE Long Island Systems, Applications, and Technology Conference (LISAT)

[7] J. S. Kumar and D. R. Patel, “A Survey on Internet of Things: Security and Privacy Issues”, IEEE Internet of Things Journal

[8] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, “An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-Defined Security”, IEEE Internet of Things Journal

[9] J. Bugeja, A. Jacobson and P. Davidson, “On Privacy and Security Challenges in Smart Connected Homes”, 2016 European Intelligence and Security Informatics Conference

[10] M. Abomhara and G. M. Køien, “Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks”, Journal of Cyber Security and Mobility

Appendix A – Implementation code for DES & 3DES Algorithms

```
#include "DES.h"
DES des;
void setup() {
  Serial.begin(9600);
  desTest();
  tdesTest();
  delay(2000);
}

void loop() {
}
void desTest() {
  byte out[8];
  byte in[] = { 0, 1, 2, 3, 4, 5, 6, 7 };
  byte key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };

  Serial.println();
  Serial.println("========= DES test ==========");

  //encrypt
  Serial.print("Encrypt...");
  unsigned long time = micros();
  des.encrypt(out, in, key);
  time = micros() - time;
  Serial.print("done. (");
  Serial.print(time);
  Serial.println(" micros)");
  printArray(out);
  //decrypt
  for (int i = 0; i < 8; i++)
  {
    in[i] = out[i];
  }
  Serial.print("Decrypt...");
  time = micros();
  des.decrypt(out, in, key);
  time = micros() - time;
  Serial.print("done. (");
  Serial.print(time);
  Serial.println(" micros)");
  printArray(out);
}
void tdesTest() {
  byte out[8];
  byte in[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
  byte key[] = {
            0x3b, 0x38, 0x98, 0x37, 0x15, 0x20, 0xf7, 0x5e, // key A
            0x92, 0x2f, 0xb5, 0x10, 0xc7, 0x1f, 0x43, 0x6e, // key B
            0x3b, 0x38, 0x98, 0x37, 0x15, 0x20, 0xf7, 0x5e, // key C
```

```
                };
  Serial.println();
  Serial.println("====== Triple-DES test ======");
  //encrypt
  Serial.print("Encrypt...");
  unsigned long time = micros();
  des.tripleEncrypt(out, in, key);
  time = micros() - time;
  Serial.print("done. (");
  Serial.print(time);
  Serial.println(" micros)");
  printArray(out);
  //decrypt
  for (int i = 0; i < 8; i++)
  {
    in[i] = out[i];
  }
  Serial.print("Decrypt...");
  time = micros();
  des.tripleDecrypt(out, in, key);
  time = micros() - time;
  Serial.print("done. (");
  Serial.print(time);
  Serial.println(" micros)");
  printArray(out);
}

void printArray(byte output[])
{
  for (int i = 0; i < 8; i++)
  {
    if (output[i] < 0x10)
    {
      Serial.print("0");
    }
    Serial.print(output[i], HEX);
    Serial.print(" ");
  }
  Serial.println();
}
```

Appendix B – Implementation code for AES Algorithm

```
#include <AES.h>
#include <AES_config.h>
#include <printf.h>
AES aes;
byte key[] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12,
0x13, 0x14, 0x15
};
byte plain[] = {
  0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37
};
int plainLength = sizeof(plain);
int padedLength = plainLength + N_BLOCK - plainLength % N_BLOCK;
unsigned long long int my_iv = 01234567;

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    yield();
  }
  printf_begin();
  delay(500);

   prekey_test();
}

void loop() {
}

void prekey(int bits) {
  //aes.iv_inc();
  byte iv[N_BLOCK];
  byte plain_p[16];
  byte cipher[16];
  byte check[16];
  unsigned long ms = micros();
  aes.set_IV(my_iv);
  aes.get_IV(iv);
  aes.do_aes_encrypt(plain, plainLength, cipher, key, bits, iv);
  ms = micros() - ms;
  Serial.print("Encryption took: ");
  Serial.println(ms);

  ms = micros();
  aes.set_IV(my_iv);
  aes.get_IV(iv);
  aes.do_aes_decrypt(cipher, padedLength, check, key, bits, iv);
  Serial.print("Decryption took: ");
  Serial.println(micros() - ms);
```

```
  Serial.print("KEY :");
  aes.printArray(key, 16);
  Serial.println();

  Serial.print("PLAIN :");
  aes.printArray(plain, (bool)true);
  Serial.println();

  Serial.print("CIPHER:");
  aes.printArray(cipher, (bool)false);
  Serial.println();

  Serial.print("CHECK :");
  aes.printArray(check, (bool)true);
  Serial.println();

  Serial.print("\nIV    :");
  aes.printArray(iv, 16);
  Serial.println();
  Serial.println();
}

void prekey_test() {
  prekey(128);
}
```

Appendix C – Implementation code for Hybrid Algorithm

```
#include <stdio.h>      // printf
#include <stdlib.h>     // exit
#include <string.h>     // memcpy, memset
#include <stdint.h>     // uint8_t

void setup() {
  Serial.begin(9600);
  uint8_t plain[16] = {0x0, 0x0, 0x0, 0x1,
                       0x0, 0x2, 0x0, 0x3,
                       0x0, 0x4, 0x0, 0x5,
                       0x0, 0x6, 0x0, 0x7};
  uint8_t inp_key[32] = {0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0,
                      0x0, 0x0, 0x0, 0x0};

  Serial.println("Plain Text: ");
  for(int i=0; i<16; i++) {
    Serial.print(plain[i], HEX);
    Serial.print(" ");
    if((i+1)%4 == 0)
      Serial.println();
  }

  uint8_t key[32];
  memcpy(key,inp_key,32);
  uint8_t arrkey[10][32];
  keyExpansion(key,arrkey);

  encrypt(plain, arrkey);
  decrypt(plain, arrkey);
}

void loop() {
}

// for debugging
#define DEBUG_KEY_FLAG 0
#define DEBUG_CIPHER_STATE_FLAG 0

#define DEBUG_KEY(a,b,c,d,e,f) do {         \
    if (DEBUG_KEY_FLAG) {                   \
        Serial.print(a);                    \
        Serial.print(b);                    \
        Serial.print(" -- ");               \
        Serial.print(c, HEX);               \
```

```
        Serial.print(d, HEX);               \
        Serial.print(e, HEX);               \
        Serial.println(f, HEX);             \
    }                                       \
} while(0)

#define DEBUG_CIPHER_STATE(a) do {          \
    if (DEBUG_CIPHER_STATE_FLAG) {          \
        for (int r=0; r<4; r++) {           \
          Serial.print(a);                  \
          Serial.print(state[r][0], HEX);   \
          Serial.print(state[r][1], HEX);   \
          Serial.print(state[r][2], HEX);   \
          Serial.println(state[r][3], HEX); \
        }                                   \
    }                                       \
} while(0)

uint8_t xtime(uint8_t x)
 {
     return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
 }

 #define MULTIPLY_AS_A_FUNCTION 0
 // Multiply is used to multiply numbers in the field GF(2^8)
 #if MULTIPLY_AS_A_FUNCTION
 uint8_t Multiply(uint8_t x, uint8_t y)
 {
   return (((y & 1) * x) ^
        ((y>>1 & 1) * xtime(x)) ^
        ((y>>2 & 1) * xtime(xtime(x))) ^
        ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^
        ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))));
 }
 #else
 #define Multiply(x, y)                              \
       (  ((y & 1) * x) ^                            \
       ((y>>1 & 1) * xtime(x)) ^                     \
       ((y>>2 & 1) * xtime(xtime(x))) ^              \
       ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^       \
       ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))   \

 #endif

const uint8_t s_box[16]=
{0xC,0x5,0x6,0xB,0x9,0x0,0xA,0xD,0x3,0xE,0xF,0x8,0x4,0x7,0x1,0x2};
const uint8_t inv_s_box[16]=
{0x5,0xE,0xF,0x8,0xC,0x1,0x2,0xD,0xB,0x4,0x6,0x3,0x0,0x7,0x9,0xA};

void addRoundKey(uint8_t state[16], uint8_t key[32]) {
  for(int i=0; i<16; i++) {
```

```
    state[15-i] = state[15-i] ^ key[31-i];
  }
}

void SubByte(uint8_t state[16]) {
  for(int i=0; i<16; i++) {
    state[i] = s_box[state[i]];
  }
}

void InvSubByte(uint8_t state[16]) {
    for(int i=0; i<16; i++) {
    state[i] = inv_s_box[state[i]];
  }
}

void shiftRows(uint8_t state[16]) {
  uint8_t temp;

  temp = state[4];
  state[4] = state[5];
  state[5] = state[6];
  state[6] = state[7];
  state[7] = temp;

  temp = state[8];
  state[8] = state[10];
  state[10] = temp;
  temp = state[9];
  state[9] = state[11];
  state[11] = temp;

  temp = state[12];
  state[12] = state[15];
  state[15] = state[14];
  state[14] = state[13];
  state[13] = temp;
}

void InvShiftRows(uint8_t state[16])  {
    uint8_t temp;
    temp = state[7];
    state[7] = state[6];
    state[6] = state[5];
    state[5] = state[4];

    state[4] = temp;

    temp = state[8];
    state[8] = state[10];
    state[10] = temp;
```

```
    temp = state[9];
    state[9] = state[11];
    state[11] = temp;

    temp = state[12];

    state[12] = state[13];
    state[13] = state[14];
    state[14] = state[15];

    state[15] = temp;

}

uint8_t multByTwo(uint8_t x) {
    return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
}

void PSub(uint8_t state[16]) {
  uint8_t temp[16];
  for(int i=0; i<15; i++)
    temp[i*4 % 15] = state[i];
  temp[15] = state[15];
  state = temp;
}

void InvPSub(uint8_t state[16]) {
  uint8_t temp[16];
  for(int i=0; i<15; i++)
    temp[i] = state[i*4 % 15];
  temp[15] = state[15];
  state = temp;
}


void mixColumns(uint8_t state[16]) {
  uint8_t temp[4];
  for(int c=0; c<4; c++) {
    temp[0] = state[4*0 + c];
    temp[1] = state[4*1 + c];
    temp[2] = state[4*2 + c];
    temp[3] = state[4*3 + c];

    state[4*0 + c] = multByTwo(temp[0]) ^ temp[1] ^ multByTwo(temp[1]) ^ temp[2] ^
temp[3];
    state[4*1 + c] = temp[0] ^ multByTwo(temp[1]) ^ temp[2] ^ multByTwo(temp[2]) ^
temp[3];
    state[4*2 + c] = temp[0] ^ temp[1] ^ multByTwo(temp[2]) ^ temp[3] ^
multByTwo(temp[3]);
    state[4*3 + c] = temp[0] ^ multByTwo(temp[0]) ^ temp[1] ^ temp[2] ^
multByTwo(temp[3]);
```

```
  }
}

void InvMixColumns(uint8_t state[16])  {
  int i;
  uint8_t a,b,c,d;
  for(i=0;i<4;++i)
  {
    a = state[4*0 + i];
    b = state[4*1 + i];
    c = state[4*2 + i];
    d = state[4*3 + i];

    state[4*0 + i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^
Multiply(d, 0x09);
    state[4*1 + i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^
Multiply(d, 0x0d);
    state[4*2 + i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^
Multiply(d, 0x0b);
    state[4*3 + i] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^
Multiply(d, 0x0e);
   }
}

void keyUpdate(uint8_t state[32]) {
    //First half left shifting
    for(int i=15; i>=2; i--) {
      state[i] = state[i-2];
    }
    state[0] = 0;
    state[1] = 0;

    //Addition & XOR
    for(int i=0; i<16; i++) {
      state[i] = state[i] + state[i+16];
      state[i] = state[i] ^ 0;
    }

    //Second half shifting
    for(int i=16; i<31; i++) {
      state[i] = state[i+1];
    }
    state[32] = 0;

    for(int i=16; i<32; i++) {
      state[i] = state[i] + state[i-16];
    }
}

void keyExpansion(uint8_t key[32], uint8_t arrkey[10][32]) {
  for(int i=0; i<11; i++) {
```

```
    memset(arrkey[i],key,32);
    keyUpdate(key);    //SPECK
  }
}

void encrypt(uint8_t state[16], uint8_t key[10][32]) {
  unsigned long t = micros();
  addRoundKey(state, key[0]); //PRESENT
  for(int RC=1; RC<11; RC++) {
       mixColumns(state);
       addRoundKey(state, key[RC]); //PRESENT
       shiftRows(state);        //AES
       PSub(state);             //PRESENT
  }
  t = micros() - t;
  Serial.print("Encryption time taken: ");
  Serial.println(t);
  Serial.println();

  /*------------------Printing--------------------*/
  Serial.println("Cipher Text: ");
  for(int i=0; i<16; i++) {
    Serial.print(state[i], HEX);
    Serial.print(" ");
    if((i+1)%4 == 0)
      Serial.println();
  }
  Serial.println();
  /*---------------End of Printing----------------*/

}

void decrypt(uint8_t state[16], uint8_t key[10][32]) {

  unsigned long t1 = micros();
  for(int RC=10; RC>=1; RC--) {
     InvPSub(state);            //PRESENT
     InvShiftRows(state);       //AES
     addRoundKey(state, key[RC]);   //PRESENT
     InvMixColumns(state);      //AES
  }

  addRoundKey(state, key[0]);   //PRESENT
  unsigned long t2 = micros();

  Serial.print("Decryption time taken: ");
  Serial.println(t2-t1);
  Serial.println();

  Serial.println("Deciphered Text: ");
  for(int i=0; i<16; i++) {
```

```
    Serial.print(state[i], HEX);
    Serial.print(" ");
    if((i+1)%4 == 0)
      Serial.println();
  }

  Serial.println();
}
```

Appendix D – Implementation code for Door Locking System

```
#include <LiquidCrystal.h>
#include <Servo.h>
#include <Keypad.h>
#include <stdio.h>      // printf
#include <stdlib.h>     // exit
#include <string.h>     // memcpy, memset
#include <stdint.h>     // uint8_t
Servo myservo;
int pos=0; // position of servo motor
LiquidCrystal lcd(A4, A5, A3, A2, A1, A0);
const byte rows=4;
const byte cols=3;
char key[rows][cols]={
                      {'1','2','3'},
                      {'4','5','6'},
                      {'7','8','9'},
                      {'*','0','#'}
                     };
byte rowPins[rows]={0,1,2,3};
byte colPins[cols]={4,5,6};
Keypad keypad= Keypad(makeKeymap(key),rowPins,colPins,rows,cols);
String password="0123";
String inp = "";
int currentposition=0;
void setup() {
  displayscreen();
  Serial.begin(9600);
  myservo.attach(9); //Servo motor connection
  lcd.begin(16,2);
}
void loop() {
  if(currentposition==0) {
    displayscreen();
  }
  int l ;
  char code=keypad.getKey();
  if(code!=NO_KEY) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("PASSWORD:");
    lcd.setCursor(7,1);
    lcd.print(" ");
    lcd.setCursor(7,1);
    for(l=0;l<=currentposition;++l) {
        lcd.print("*");
        //keypress();
    }
    if(code == '#') {
```

```
      //Here the verification is happening locally, but in case the verification
needs to happen in the cloud the encrypted input is sent.
      if(verifyPassword(inp)) {
        unlockdoor();
        currentposition = 0;
      } else {
        incorrect();
        currentposition = 0;
      }
      inp ="";
    }
    else {
      inp += String(code);
      ++currentposition;
    }
  }
}
void expand(String inp, uint8_t plain[16]) {
  for(int i=0; i<inp.length(); i++) {
    plain[i] = uint8_t(inp[i])-uint8_t('0');
  }
  for(int i=inp.length(); i<16; i++) {
    plain[i] = 0xA;
  }
}
String convertToString(uint8_t plain[16]) {
  String res = "";
  for(int i=0; i<16; i++) {
    if(plain[i] != 0xA)
      res += String(plain[i]);
  }
  return res;
}
bool verifyPassword(String inp) {
      uint8_t plain[16];
      expand(inp, plain);

       //Add the key part here
      uint8_t inp_key[32] = {0x0, 0x1, 0x0, 0x2,
                     0x0, 0x4, 0x0, 0x5,
                     0x0, 0x6, 0x0, 0x7,
                     0x0, 0x8, 0x0, 0x9,
                     0x1, 0x0, 0x1, 0x1,
                     0x1, 0x2, 0x1, 0x3,
                     0x1, 0x4, 0x1, 0x5};
      uint8_t key[32];
      memcpy(key,inp_key,32);
      uint8_t arrkey[10][32];
      keyExpansion(key,arrkey);
      encrypt(plain, arrkey);
      //Cloud code goes here
```

```
      decrypt(plain, arrkey);
  String s = convertToString(plain);
  if(s == password) {
    return true;
  }
  return false;
}
//------------------ Function 1- OPEN THE DOOR--------------//
void unlockdoor() {
  delay(900);
  lcd.setCursor(0,0);
  lcd.println(" ");
  lcd.setCursor(1,0);
  lcd.print("Access Granted");
  lcd.setCursor(4,1);
  lcd.println("WELCOME!!");
  lcd.setCursor(15,1);
  lcd.println(" ");
  lcd.setCursor(16,1);
  lcd.println(" ");
  lcd.setCursor(14,1);
  lcd.println(" ");
  lcd.setCursor(13,1);
  lcd.println(" ");
  // open the door
  for(pos = 180; pos>=0; pos-=5) {
    myservo.write(pos);
    delay(5);
  }

  delay(2000);
  delay(1000);
  counterbeep();
  delay(1000);
  // close the door
  for(pos = 0; pos <= 180; pos +=5) {
  // in steps of 1 degree
    myservo.write(pos);
    delay(15);
    currentposition=0;
    lcd.clear();
    displayscreen();
  }
}
//--------------------Function 2- Wrong code--------------//
void incorrect() {
  delay(500);
  lcd.clear();
  lcd.setCursor(1,0);
  lcd.print("CODE");
  lcd.setCursor(6,0);
```

```
  lcd.print("INCORRECT");
  lcd.setCursor(15,1);
  lcd.println(" ");
  lcd.setCursor(4,1);
  lcd.println("GET AWAY!!!");

  lcd.setCursor(13,1);
  lcd.println(" ");
  Serial.println("CODE INCORRECT YOU ARE UNAUTHORIZED");
  delay(3000);
  lcd.clear();
  displayscreen();
}
//-------Function 3 - CLEAR THE SCREEN--------------------/
void clearscreen() {
  lcd.setCursor(0,0);
  lcd.println(" ");
  lcd.setCursor(0,1);
  lcd.println(" ");
  lcd.setCursor(0,2);
  lcd.println(" ");
  lcd.setCursor(0,3);
  lcd.println(" ");
}
//------------Function 4 - DISPLAY FUNCTION--------------------//
void displayscreen() {
  lcd.setCursor(0,0);
  lcd.println("*ENTER THE CODE*");
  lcd.setCursor(1 ,1);
  lcd.println("TO OPEN DOOR!!");
}
//--------------Function 5 - Count down------------------//
void counterbeep()
{
  delay(1200);
  lcd.clear();
  lcd.setCursor(2,15);
  lcd.println(" ");
  lcd.setCursor(2,14);
  lcd.println(" ");
  lcd.setCursor(2,0);
  delay(200);
  lcd.println("GET IN WITHIN:::");
  lcd.setCursor(4,1);
  lcd.print("5");
  delay(200);
  lcd.clear();
  lcd.setCursor(2,0);
  lcd.println("GET IN WITHIN:");
  delay(1000);
  lcd.setCursor(2,0);
```

```
lcd.println("GET IN WITHIN:");
lcd.setCursor(4,1); //2
lcd.print("4");
delay(100);
lcd.clear();
lcd.setCursor(2,0);
lcd.println("GET IN WITHIN:");
delay(1000);

lcd.setCursor(2,0);
lcd.println("GET IN WITHIN:");
lcd.setCursor(4,1);
lcd.print("3");
delay(100);
lcd.clear();
lcd.setCursor(2,0);
lcd.println("GET IN WITHIN:");
delay(1000);
lcd.setCursor(2,0);
lcd.println("GET IN WITHIN:");
lcd.setCursor(4,1);
lcd.print("2");
delay(100);
lcd.clear();
lcd.setCursor(2,0);
lcd.println("GET IN WITHIN:");
delay(1000);
lcd.setCursor(4,1);
lcd.print("1");
delay(100);
lcd.clear();
lcd.setCursor(2,0);
lcd.println("GET IN WITHIN::");
delay(1000);
delay(40);
lcd.clear();
lcd.setCursor(2,0);
lcd.print("RE-LOCKING");
delay(500);
lcd.setCursor(12,0);
lcd.print(".");
delay(500);
lcd.setCursor(13,0);
lcd.print(".");
delay(500);
lcd.setCursor(14,0);
lcd.print(".");
delay(400);
lcd.clear();
lcd.setCursor(4,0);
lcd.print("LOCKED!");
```

```
  delay(440);
}
//--------------Hybrid Algorithm Section-----------------//
// for debugging
#define DEBUG_KEY_FLAG 0
#define DEBUG_CIPHER_STATE_FLAG 0
#define DEBUG_KEY(a,b,c,d,e,f) do {         \
    if (DEBUG_KEY_FLAG) {                   \
        Serial.print(a);                    \
        Serial.print(b);                    \
        Serial.print(" -- ");               \
        Serial.print(c, HEX);               \
        Serial.print(d, HEX);               \
        Serial.print(e, HEX);               \
        Serial.println(f, HEX);             \
    }                                       \
} while(0)
#define DEBUG_CIPHER_STATE(a) do {          \
    if (DEBUG_CIPHER_STATE_FLAG) {          \
        for (int r=0; r<4; r++) {           \
          Serial.print(a);                  \
          Serial.print(state[r][0], HEX);   \
          Serial.print(state[r][1], HEX);   \
          Serial.print(state[r][2], HEX);   \
          Serial.println(state[r][3], HEX); \
        }                                   \
    }                                       \
} while(0)
uint8_t xtime(uint8_t x)
 {
     return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
 }
 #define MULTIPLY_AS_A_FUNCTION 0
 // Multiply is used to multiply numbers in the field GF(2^8)
 #if MULTIPLY_AS_A_FUNCTION
 uint8_t Multiply(uint8_t x, uint8_t y)
 {
   return (((y & 1) * x) ^
        ((y>>1 & 1) * xtime(x)) ^
        ((y>>2 & 1) * xtime(xtime(x))) ^
        ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^
        ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))));
 }
 #else
 #define Multiply(x, y)                              \
       (  ((y & 1) * x) ^                            \
       ((y>>1 & 1) * xtime(x)) ^                     \
       ((y>>2 & 1) * xtime(xtime(x))) ^              \
       ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^       \
       ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))   \
```

```
 #endif
const uint8_t s_box[16]=
{0xC,0x5,0x6,0xB,0x9,0x0,0xA,0xD,0x3,0xE,0xF,0x8,0x4,0x7,0x1,0x2};
const uint8_t inv_s_box[16]=
{0x5,0xE,0xF,0x8,0xC,0x1,0x2,0xD,0xB,0x4,0x6,0x3,0x0,0x7,0x9,0xA};
void addRoundKey(uint8_t state[16], uint8_t key[32]) {
  for(int i=0; i<16; i++) {
    state[15-i] = state[15-i] ^ key[31-i];
  }
}
void SubByte(uint8_t state[16]) {
  for(int i=0; i<16; i++) {
    state[i] = s_box[state[i]];
  }
}
void InvSubByte(uint8_t state[16])
{
    for(int i=0; i<16; i++) {
    state[i] = inv_s_box[state[i]];
  }
}
void shiftRows(uint8_t state[16]) {
  uint8_t temp;
  temp = state[4];
  state[4] = state[5];
  state[5] = state[6];
  state[6] = state[7];
  state[7] = temp;
  temp = state[8];
  state[8] = state[10];
  state[10] = temp;
  temp = state[9];
  state[9] = state[11];
  state[11] = temp;
  temp = state[12];
  state[12] = state[15];
  state[15] = state[14];
  state[14] = state[13];
  state[13] = temp;
}

void InvShiftRows(uint8_t state[16])
{
    uint8_t temp;
    temp = state[7];
    state[7] = state[6];
    state[6] = state[5];
    state[5] = state[4];
    state[4] = temp;
    temp = state[8];
    state[8] = state[10];
```

```
    state[10] = temp;
    temp = state[9];
    state[9] = state[11];
    state[11] = temp;
    temp = state[12];
    state[12] = state[13];
    state[13] = state[14];
    state[14] = state[15];
    state[15] = temp;
}
uint8_t multByTwo(uint8_t x) {
    return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
}
void PSub(uint8_t state[16])
{
  uint8_t temp[16];
  for(int i=0; i<15; i++)
    temp[i*4 % 15] = state[i];
  temp[15] = state[15];
  state = temp;
}
void InvPSub(uint8_t state[16])
{
  uint8_t temp[16];
  for(int i=0; i<15; i++)
    temp[i] = state[i*4 % 15];
  temp[15] = state[15];
  state = temp;
}
void mixColumns(uint8_t state[16]) {
  uint8_t temp[4];
  for(int c=0; c<4; c++) {
    temp[0] = state[4*0 + c];
    temp[1] = state[4*1 + c];
    temp[2] = state[4*2 + c];
    temp[3] = state[4*3 + c];

    state[4*0 + c] = multByTwo(temp[0]) ^ temp[1] ^ multByTwo(temp[1]) ^ temp[2] ^
temp[3];
    state[4*1 + c] = temp[0] ^ multByTwo(temp[1]) ^ temp[2] ^ multByTwo(temp[2]) ^
temp[3];
    state[4*2 + c] = temp[0] ^ temp[1] ^ multByTwo(temp[2]) ^ temp[3] ^
multByTwo(temp[3]);
    state[4*3 + c] = temp[0] ^ multByTwo(temp[0]) ^ temp[1] ^ temp[2] ^
multByTwo(temp[3]);
  }
}

void InvMixColumns(uint8_t state[16])
{
  int i;
```

```
  uint8_t a,b,c,d;
  for(i=0;i<4;++i)
  {
    a = state[4*0 + i];
    b = state[4*1 + i];
    c = state[4*2 + i];
    d = state[4*3 + i];

    state[4*0 + i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^
Multiply(d, 0x09);
    state[4*1 + i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^
Multiply(d, 0x0d);
    state[4*2 + i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^
Multiply(d, 0x0b);
    state[4*3 + i] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^
Multiply(d, 0x0e);
   }
}
void keyUpdate(uint8_t state[32]) {
    //First half left shifting
    for(int i=15; i>=2; i--) {
      state[i] = state[i-2];
    }
    state[0] = 0;
    state[1] = 0;
    //Addition & XOR
    for(int i=0; i<16; i++) {
      state[i] = state[i] + state[i+16];
      state[i] = state[i] ^ 0;
    }
    //Second half shifting
    for(int i=16; i<31; i++) {
      state[i] = state[i+1];
    }
    state[32] = 0;

    for(int i=16; i<32; i++) {
      state[i] = state[i] + state[i-16];
    }
}
void keyExpansion(uint8_t key[32], uint8_t arrkey[10][32]) {
  for(int i=0; i<11; i++) {
    memset(arrkey[i],key,32);
    keyUpdate(key);    //SPECK
  }
}
void encrypt(uint8_t state[16], uint8_t key[10][32]) {
  addRoundKey(state, key[0]); //PRESENT
  for(int RC=1; RC<11; RC++) {
       mixColumns(state);
       addRoundKey(state, key[RC]); //PRESENT
```

```
      shiftRows(state);        //AES
      PSub(state);             //PRESENT
  }
}
void decrypt(uint8_t state[16], uint8_t key[10][32]) {
  for(int RC=10; RC>=1; RC--) {
     InvPSub(state);           //PRESENT
     InvShiftRows(state);      //AES
     addRoundKey(state, key[RC]);   //PRESENT
     InvMixColumns(state);     //AES
  }
  addRoundKey(state, key[0]);   //PRESENT
}
```