CHAPTER 4
ANALYSIS AND DESIGN

4.1. Analysis

This research uses Arduino IDE as the main software and uses C language to make the encryption, decryption, and compression programs. The module required to do this research is included the followings item :

1. Arduino UNO
2. Bread Board
3. Jumper cable
4. RFID reader module
5. RFID card or tag

Arduino is used to execute the RSA, AES, and compression algorithm. The plaintext would be converted into ciphertext with or without compression and stored inside the RFID card or tag memory block.

4.1.1. RSA, AES, and Modified Huffman Compression in Arduino

There are two types of cryptography algorithms based on the respective type of key: RSA (Asymmetric-key) and AES (Symmetric-key). The data inserted inside the RFID card or tag memory would be compressed and encrypted using the selected algorithms before.

For RSA implementation, this project will use a modified program inspired by the tutorial source http://www.trytoprogram.com/cpp-examples/cplusplus-program-encrypt-decrypt-string/. In the original tutorial, the program uses C++. When executed in Arduino, the program will show an error message because not all functions from the C++ library are compatible with Arduino IDE language. The message will be converted from string to char for easier looping. Also the output of the encrypted message will be shown in hexadecimal format for the insertion of data to the RFID card or tag memory block.

For AES implementation, this project is using a library from source https://github.com/DavyLandman/AESLib. The AES plaintext will be encrypted at 16 bytes, so even when the data input is not 16 bytes in character, the encrypted plaintext will be outputting 16 bytes. However, when the plaintext is over 16 bytes, the rest of the data would not be encrypted appropriately or lost. Therefore, the requirement to implement this AES library is 16 bytes of characters as the key.

4.1.2. MFRC522 RFID Reader and Tag

The RFID reader module used the 13.56MHz frequency, which is an industrial standard (ISM) band. Therefore, the RFID card or tag data could be read by the module when in the range of approximately 5 cm and only if the card or tag operates on 13,56 Mhz. Furthermore, the MFRC522 reader module has the following pin configuration and description (Figure 4.1) :

RC522 Pin Configuration

Pin Number	Pin Name	Description
1	Vcc	Used to Power the module, typically 3.3V is used
2	RST	Reset pin – used to reset or power down the module
3	Ground	Connected to Ground of system
4	IRQ	Interrupt pin – used to wake up the module when a device comes into range
5	MISO/SCL/Tx	MISO pin when used for SPI communication, acts as SCL for I2c and Tx for UART.
6	MOSI	Master out slave in pin for SPI communication
7	SCK	Serial Clock pin – used to provide clock source
8	SS/SDA/Rx	Acts as Serial input (SS) for SPI communication, SDA for IIC and Rx during UART

Figure 4.1 MFR522 Pin Configuration and Description (source : https://components101.com/wireless/rc522-rfid-module)

Meanwhile, the RFID card or tag uses MIFARE classic with 1 KB of memory, and it offers 1024 bytes of storage. The memory splits into 16 sectors; each sector has four blocks and each block could store 16 bytes of data. Figure 4.2 below will give a more detailed look at how the MIFARE card memory is constructed.

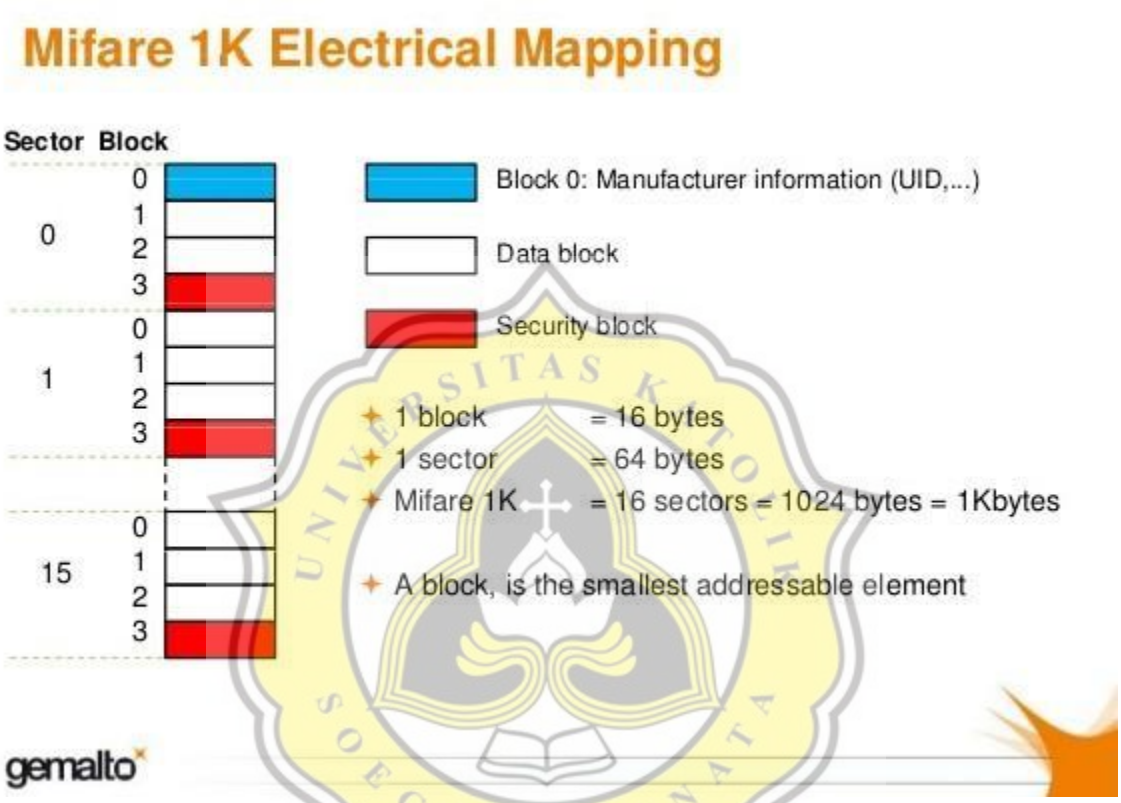

Figure 4.2 MIFARE RFID Tag Memory Construction (source : https://www.slideshare.net/GlenCifuentesToro/mifare-cards)

4.1.3. Encryption and Compression Speed

The encryption, compression, and decryption speed measurement are done inside the RSA, AES, and compression program using the micros() function from the Arduino IDE library. There are three calculations of time used to complete the process. The first is time to complete encryption, decrypt the encrypted text, and complete compression on a certain number of words. The result from the function micros() is in microseconds. For comparison, note that 1 micros are equal to 0.000001 seconds.

4.2. Desain

4.2.1. Data Encryption and Decryption in Arduino

Figures 4.3 and Figure 4.4 below show the flowchart of RSA and AES implementation of data encryption and decryption in Arduino.

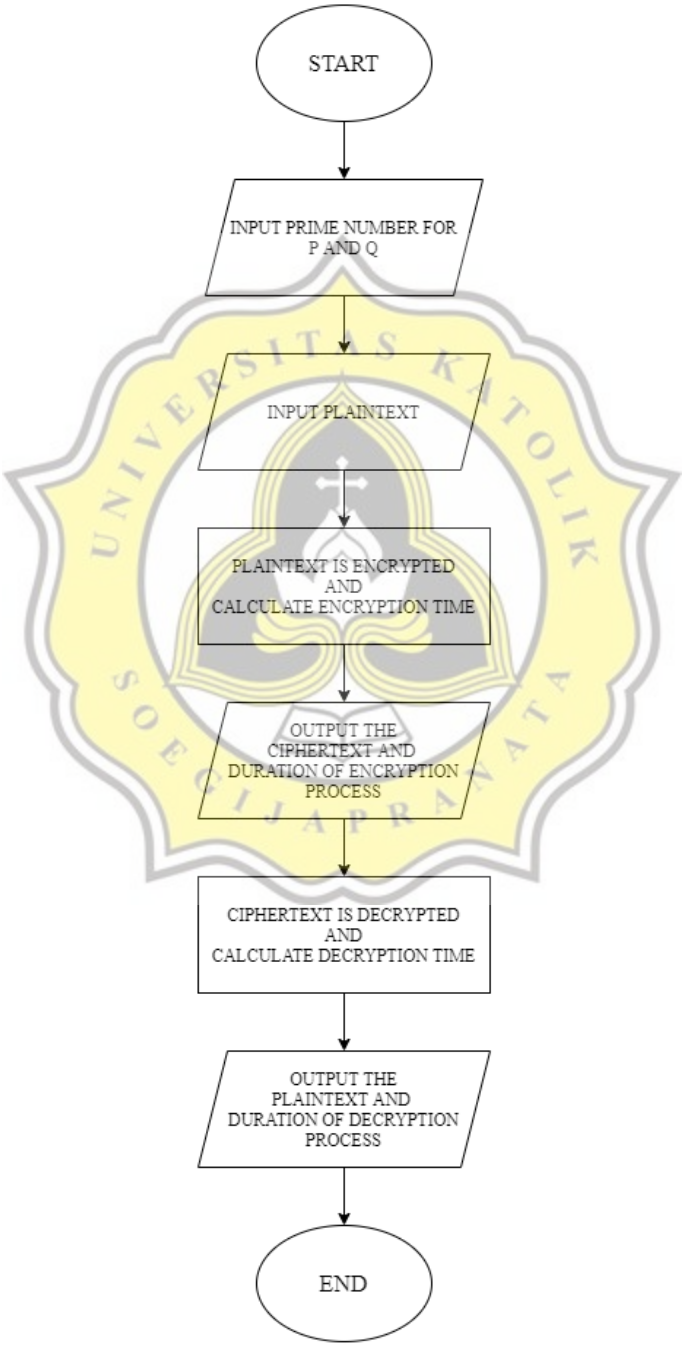

Figure 4.3 Flowchart of RSA Program

In the RSA program (Figure 4.3), the first step is to input prime numbers for variables P and Q, and this step is to generate a public and private key. Then, inputted plaintext will be converted from string to char for the encryption process. Thus, there are two calculation processes in this program: time for the encryption and decryption processes. The encryption and decryption process time can be seen in the serial monitor in microseconds as comparison material.

Figure 4.4 Flowchart of AES Program

In the AES program (Figure 4.4), the first step is to input the AES key and plaintext desired to be encrypted. Because this program used 128-bit AES, the key array's length should be exactly 16 bytes. Also, the AESLib library's instruction required that the plaintext to encrypt should be a maximum of 16 bytes, and if the text is shorter than 16 characters, the output of the encryption process will be 16 characters still. The encrypted text appears in string and hexadecimal forms output. The time calculation of this encryption and decryption also could be seen in the serial monitor in a microsecond.

4.2.2. Modified Huffman Compression in Arduino

Here is the flowchart of the data compression using the modified Huffman program in Arduino.

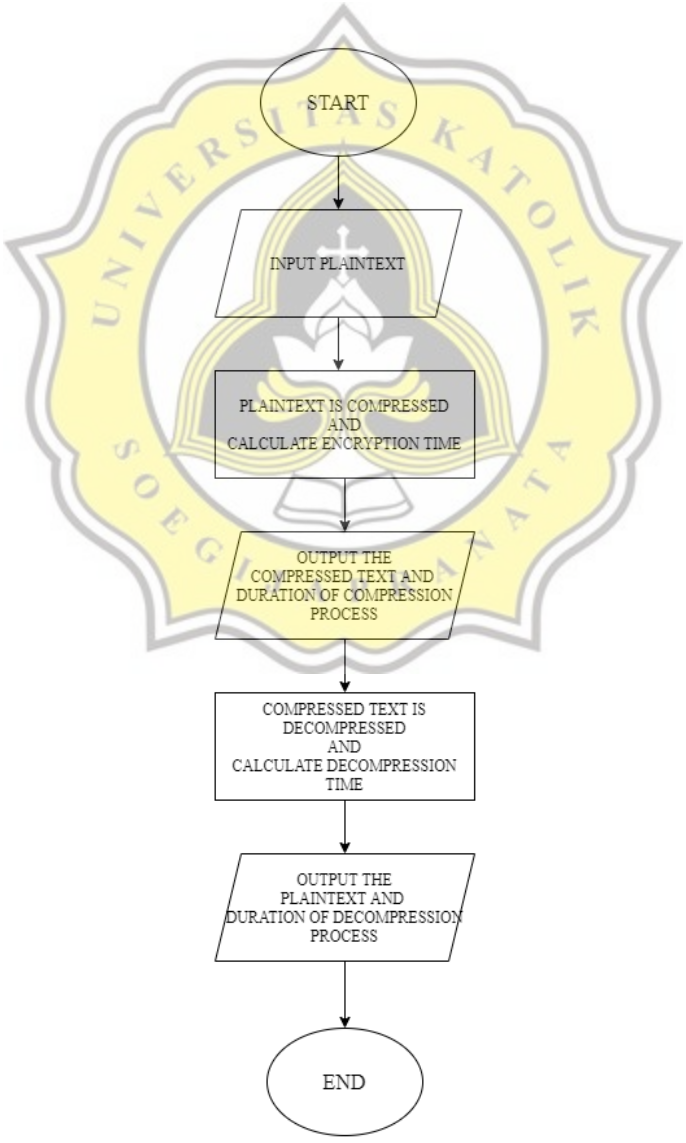

Figure 4.5 Flowchart of Modified Huffman Program

Data compression is helpful to reduce the redundancy of input data that will be encrypted. However, in this implementation, the Huffman suited for Arduino did not follow the original method with a binary code compression scheme. With the limitation of the Arduino IDE library, this program is limited to compressing a word with repeated characters in the alphabet only. This program testing without and with the combination of encryptions programs to compare the performance effectiveness.

4.2.3. Complete Arduino Module Scheme

Figure 4.6 below is the Arduino circuit scheme made for this research.

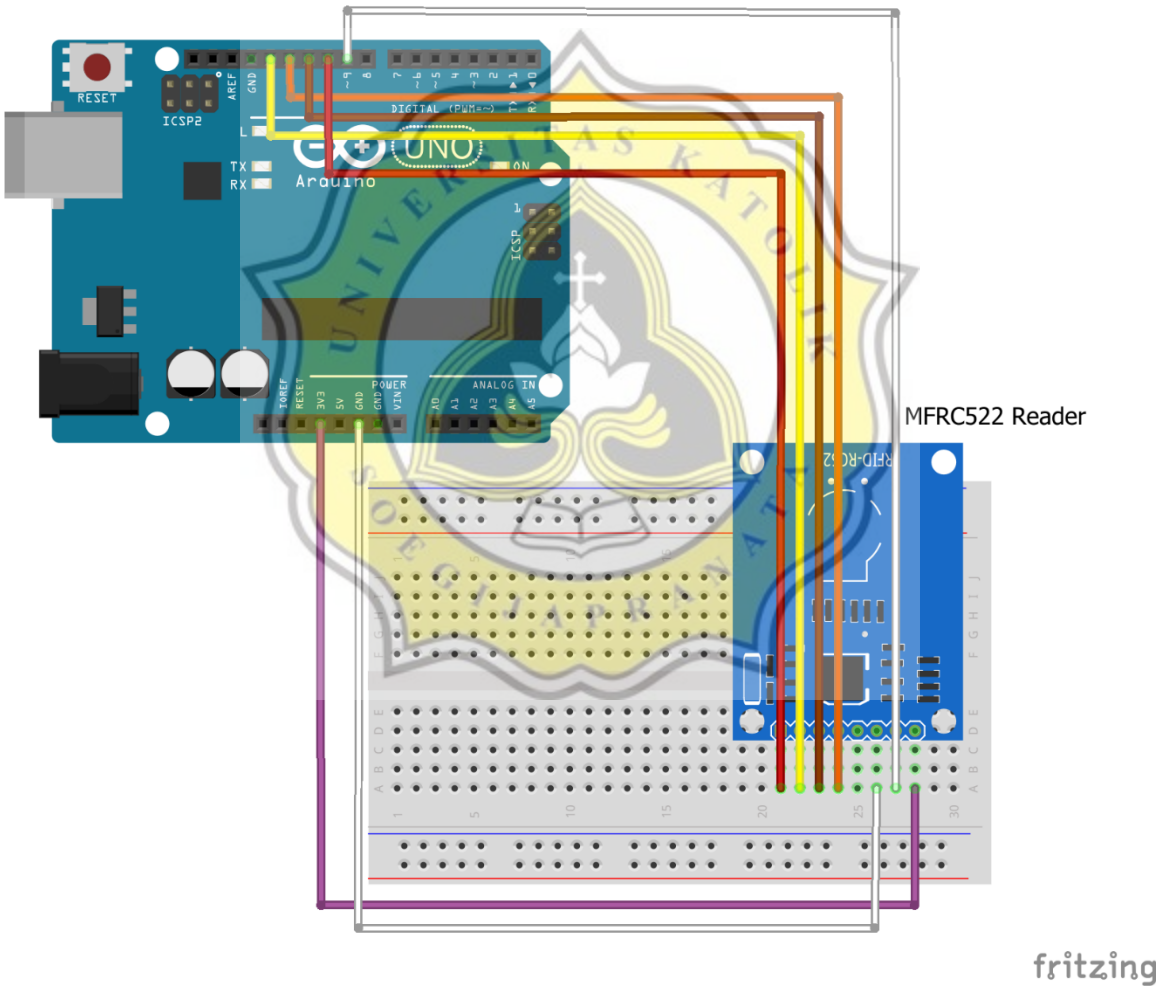

Figure 4.6 Complete Arduino Module Scheme

The encryption, decryption, and compression are processed on Arduino and the process's output is written into RFID card or tag memory using the Arduino program. The first step is to send text data. Then, whether it is uncompressed or compressed using the Modified Huffman

compression program, the output string is encrypted using RSA or AES program. After it, the output text is encrypted. Finally, the text must convert into hexadecimal to input in an RFID card or tag memory block. Note, the program will be running within the limitation of Arduino. Because of this limitation, its functionality will separate the program. All the results from the program running on the Arduino will appear in the serial monitor to compare process time and processed text result.