CHAPTER 3
RESEARCH METHODOLOGY

3.1. Rivest-Adleman-Shamir (RSA) Algorithms

Rivest-Adleman-Shamir (RSA) is categorized as an asymmetric algorithm. Although this type of encryption needed two different keys, a public key and a private key, the two keys should not be the same. Therefore, the RSA algorithm consisted of three steps, first generating a private key and public key, which was necessary to encrypt and decrypt the data. The second is the encryption process, where the plaintext is converted into ciphertext, then the last step is the decryption process which converts the ciphertext back to plaintext. More detailed steps of this algorithm explained below :

3.1.1. Key Generation Algorithm :

1. First, insert the desired two prime numbers called P and Q, which P and Q are prime numbers, but P and Q must not be the same number.

2. Then, compute the modulus N = P*Q—the value of N, later used as the modulus for private keys and public keys.

3. Calculate f(n) = (P-1)*(Q-1) to determine value of E

4. Choose an integer as E such that E is coprime of f(n) and 1 < E < f(n). The integer E is used to determine the public key and private key.

5. Compute $D = E^{-1} \bmod (f(n))$

From algorithm above concluded that the public key is (E, N) and the private key is (D, N)

3.1.2. Encryption :

1. First, obtain the value of the public key (E, N)

2. Insert the message, convert the message into positive integer M.

3. Then compute ciphertext, $C = M^E \pmod N$

4. Store the ciphertext to be decrypted later.

3.1.3. Decryption :

1. First, obtain the private key (D, N) value to decrypt the ciphertext.

2. Compute $M = C^d \pmod N$ to convert ciphertext back to plaintext.

To give an idea of how RSA algorithm works, see the example below :

1. Select prime numbers as P and Q, suppose the P = 11 and Q = 17, with E = 3.

2. Calculate N = P*Q = 11 * 17

= 187

3. Calculate f(n) = (P-1)*(Q-1) = 10 * 16

=160

4. Calculate the value of $D = \frac{1+(k \times 160)}{3}$

D = 107

5. The value of E and N becomes public key, E = 3 and N = 187.

For a plaintext example, suppose the sender message is " HELLO" and since the N is a small number, the sender would have to send this message character by character.

1. The letter 'H' in ASCII is 72, so the plaintext message would be M = 72

2. To encrypt calculate $M^E = 72^3 = 183 \pmod{187}$, the ciphertext would be C = 183

3. The sender then decrypt the ciphertext using D an N as a private key

4. To decrypt the ciphertext, calculate $C^D = 183^{107} = 72 \pmod{187}$, so the message M is 72

5. The receiver then converted 72 back to the letter 'H'.

3.2. Advanced Encryption Standard (AES) Algorithms

Advanced Encryption Standard (AES) is an algorithm that supports various block lengths and key sizes with multiple 32 bits. However, there are only three types of keys which are 128 bits, 192 bits, and 256 bits, that become the standard of AES. The difference between the three types is that the AES 128 bits used ten rounds, the AES 192 bits used 12 rounds, and the AES 256 bits used 14

rounds. This project used 128 bits AES, the 128 bits of plaintext block, or 16 bytes of data arranged into a 4 x 4 bytes array.

First, convert the 128-bit key into hexadecimal, then enter it into the 4x4 matrix of bytes array called State array. Next, the 16 bytes of plaintext will be converted into hexadecimal and inserted into the four columns by four rows matrix. The first nine rounds of operations will do the same: adding a round key, substitute bytes, shift rows and mix columns. The last round would do the operation slightly differently because it does not remix columns. More detailed steps of the AES algorithm explained below :

3.2.1. Adding Round Key

Add round key step is an XOR operation performed between states to combine cipher key and round key using Rijndael key schedule algorithm.

Figure 3.1 Adding Round Key (source : http://klinikinformatikacyber.blogspot.com/2016/03/pengertian-dan-sistem-kerja-aes-iii.html)

In the image above, the left image is ciphertext, and the right one is the round key. Thus, the process of XOR-ing is done per column. The first ciphertext column XOR-ed with the first round key column and so on.

3.2.2. Substitutes Bytes

Sub byte step is where each byte is replaced with another byte retrieved from the substitution table (S-box) accordingly. The S-box could be inverted to do the decryption process later.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3.2 S-box Table (source : http://klinikinformatikacyber.blogspot.com/2016/03/pengertian-dan-sistem-kerja-aes-iii.html)

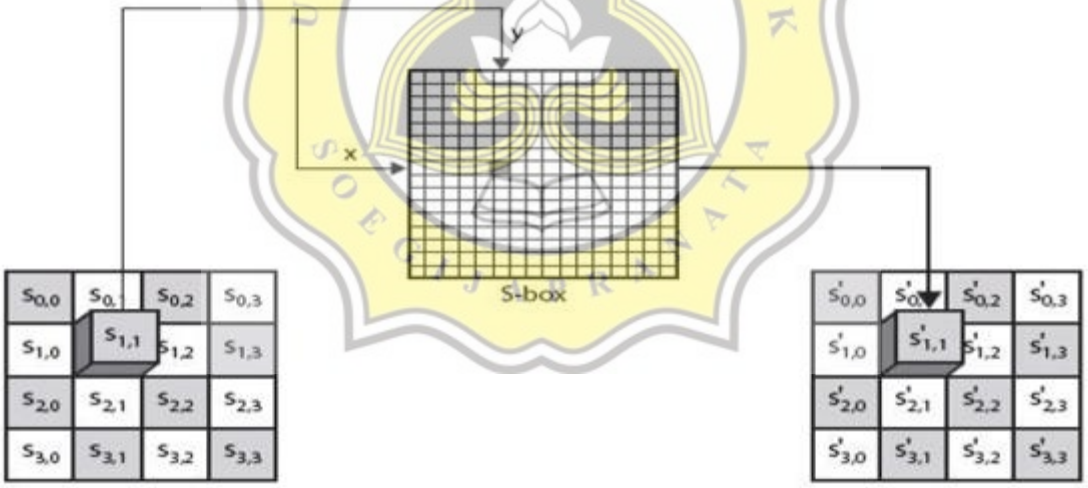

Figure 3.3 Substitution Bytes (source : http://klinikinformatikacyber.blogspot.com/2016/03/pengertian-dan-sistem-kerja-aes-iii.html)

In the image of sub bytes above, the steps are to take the content inside the matrix box and match it with the row on the left and the column on the right. From the table of S-box, we could retrieve the table by knowing which columns and rows. The last step in this operation is to transform the

entire block of a cipher into a new block with the content being the result of the steps mentioned before.

3.2.3. Shift Rows

The shift rows step is an altering step that shifts each row of the state repeatedly to a certain number of steps. The shifting process could be defined as following formulation (1) :

$$S'r, c \;=\; Sr, \Big(\big(c + shift(r, 4) \big) mod\ 4 \Big) \;\#(1)$$

The shift on each content of table or block is done per row, the first row isn't shifted, the second row is shifted by one byte, the third row is shifted by two bytes, and the fourth row is shifted by three bytes. The shift done in the block is a shifting process of the element to the left depending on how many bytes it displaced, every one byte means the element will be shifted left for one time.

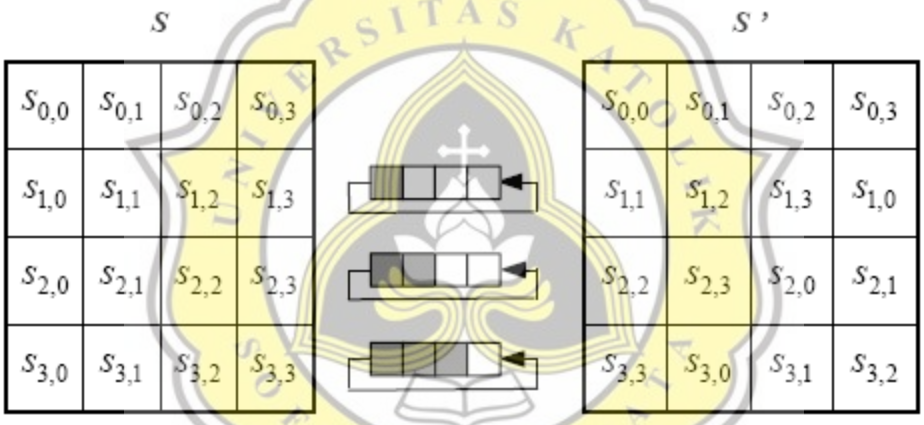

Figure 3.4 Shift Rows (source : https://www.researchgate.net/figure/AES-ShiftRows-function-taken-from-8_fig4_265112905)

3.2.4. Mix Columns

Mix columns are where the state columns are combining the four bytes inside of each column. Then, the table is multiplied using the usual matrix multiplication principle. Finally, the multiplication of the cipher block and the table, as shown in Figure 3.4 using a dot product, will be inserted into a new cipher block. The illustration in Figure 3.5 is explaining how multiplying works. That was the whole primary process of AES encryption.

02	01	01	03
03	02	01	01
01	03	02	01
01	01	02	03

Figure 3.5 Mix Columns (source : http://kriptografijaringan.blogspot.com/2016/03/enkripsi-algoritma-aes-advanced.html)

Figure 3.6 Mix Column Multiplication (source : http://kriptografijaringan.blogspot.com/2016/03/enkripsi-algoritma-aes-advanced.html)

3.2.6. AES Encryption and Decryption Flow Diagram

The following figure illustration is describing the process before in an image. First, the encryption and decryption process starts with the Adding Round Key process in the array of states, which provides security. Then, while the rest of the rounds follow in an identical series of processes where each of the rounds is doing Substitutes Byte, Shift Rows, Mix Columns, and Adding Round Key. However, in the final round, the process is slightly different because it is skipping the Mixing Columns step.

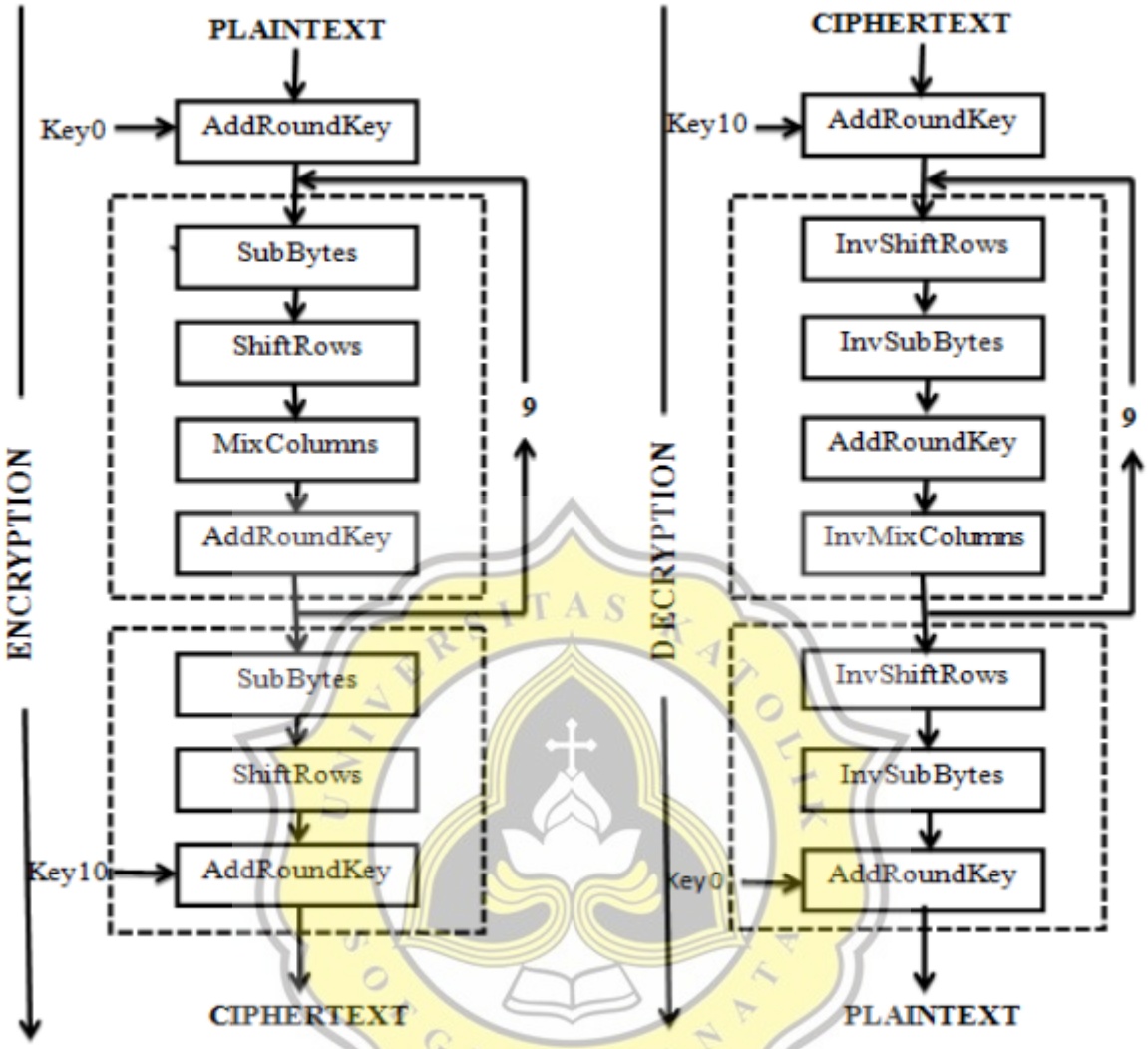

Figure 3.7 AES Encryption And Decryption Flowchart

(source : Dr. Prerna Mahajan, A. S. (2013). A Study of Encryption Algorithms AES, DES, and RSA for Security. Global Journal of Computer Science and Technology. https://computerresearch.org/index.php/computer/article/view/272)

3.3. Huffman Encoding

The Huffman method is one of many compression techniques utilizing coding in the form of bits to represent character data.

The compression method for this technique is as follows :

1. First, count the number of character types and the number of each character contained in a file.

2. Then, arrange each type of character with the order from the least amount of character to the most amount of character shown up.

3. Create a binary tree based on the sequence, from the least amount of number to the most amount of character showed up, and assign a code for each character.

4. Replace the existing data with bit codes based on the binary tree.

5. Stores the largest bit code number, then the type of characters sorted from the largest to the smallest output frequency, along with the data converted into bit code as the result of compression.

For an example of a compression technique using the Huffman method, suppose inside the text file there are characters "AAAABBBCCCCCD", this file has a size of 13 bytes because one character is equal to 1 byte.

Based on the method before, compression is do as follows :

1. Take note of the existing character and the number of characters showing up in the frequency.

A = 4, B = 3, C = 5, D = 1

2. Then, sort the character from the least number to the most, which are :

D → B → A → C

3. Create a binary tree based on the sequence of characters with the least number of frequencies to the most.

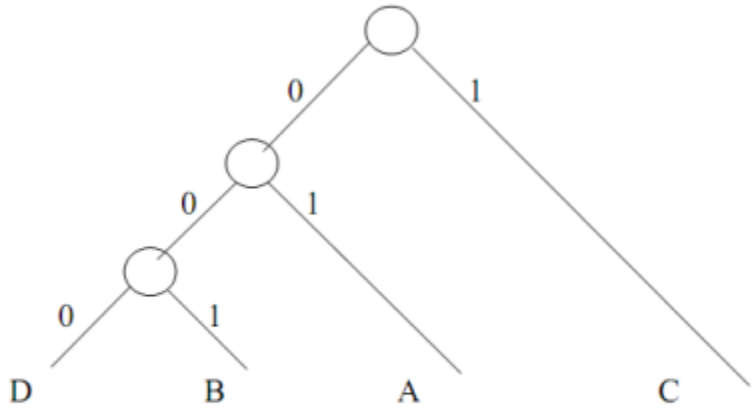

Figure 3.8 Binary Tree of 'AAAABBBCCCCCD' (source : https://barafirman.wordpress.com/write-zone/)

4. Replaces the existing data with bit codes based on the generated binary tree shown in Figure 3.7. Substitution of the character into binary code, determined from the top node or called root node :

A = 01, B = 001, C = 1, D = 000

Furthermore, based on binary code from each of these characters, all characters in the file can be changed to :

01010101001001001111110001111111

Because the number 0 and 1 represent 1 bit, so the data bit above consists of 32 bits or 4 bytes (1 byte = 8 bit).

5. Stores the bit code of the character with the largest frequency, the type of character contained in the file, and the text file data that has been encoded.

The way to store character type data is to sort the character type data from the most frequent to the least frequency into :

[C, A, B, D]

The text file above, after compressed, has a size of 1 + 4 + 4 = 9 bytes. This number consists of 1 byte of character code with the lowest frequency, four types of characters equals 4 bytes, and the 4 bytes of all character code data.

Any compressed data shall reconstruct according to the original data. Reconstructing data is better known as a data decompression method. For example, the Huffman decompression method is returning binary-coded data to a text file.

The decompression method for returning the compressed data to the original data is as follows:

1. Read the first data, which is the bit code of the last character data. This first data has a varying number of bits (in the example above, this data is equal to 1 byte) and is used as a comparison to find out whether the reconstructed character data is the last character data or not. This data always has a value equal to zero.

2. Read the compressed binary code data bit by bit. If the value of the first bit is equal to "0", then it is continued on the second bit. If the second bit also has a value of "0", it will continue reading the bits until its value equals "1".

After finding the bit value "1", all bits read are code of a character.

3. Convert a binary code into a character by counting the number of bits in the code. For example, suppose the number of bits is **n**, then the above binary code represents the **n^{th}** character in the character listing.

4. Then, repeat steps two and three until the last binary bit code. To determine whether the binary code on reading represents the last character in the character listing or not, compare all the read "0" bits with the binary code of the last character.

3.4. RSA and AES Implementation on Arduino

The main focus of this research is to implement and compare the performance of the RSA and AES encryption algorithm on Arduino with C language. Using a variable that stores plaintext (in

the form of a String) and then ciphertext converted into hexadecimal stored inside RFID memory block.

3.5. Modified Huffman Implementation on Arduino

To optimize the encryption used above, before the plaintext is encrypted, the text would be shortened or compressed using this method. Although implementation of the Huffman method for this research is slightly different from the original method, it is adapted to the use case. In this project, to reduce data redundancy and reduce the time for encryption and decryption, data compression uses a simplified version of the Huffman compression method. This compression method called Run Length Encoding is only effective when the text includes the same character that is repeated in the same order of each character (for example: 'AAAABBBCCCCDD'). If there is a unique character inside the plaintext, the compression will output the same character.

3.6. Compare RSA and AES Encryption without Compression Speeds

The performance of data encryption in RSA and AES is done on separate programs using a micro (ms) time unit. This research measures encryption and decryption times using the Arduino function.

3.7. Compare the RSA and AES Encryption with Compression Speeds

To determine whether the performance of encryption speed using compression techniques affects the time to finish or not.

3.8. Testing

After installing all the modules, the testing phase follows next. After the test, compare the time used for RSA encryption, AES encryption, and Compression Method. These research results are presented in this project report.