Mirror Cipher using Feistel Network

Ihsan Muhammad Asnadi[1] **Ranindya Paramitha**[2] **Tony**[3]

[123] Informatics Department, Institut Teknologi Bandung, Bandung 40132, Indonesia
E-mail: [1]13516028@std.stei.itb.ac.id [2]13516006@std.stei.itb.ac.id [3]13516010@std.stei.itb.ac.id

Abstract. Mirror cipher is a cipher built by creativity which has a specific feature of mirrored round function. As other ciphers, mirror cipher could be used to secure messages' confidentiality and integrity. This cipher receives message and key inputs from its user. Then, it runs 9 rounds of feistel networks in ECB modes. Each round would run a round function which consists of 5 functions in mirrored order (9 function calls in total): s-box substitution, row substitution, column substitution, column cumulative xor, and round key addition. This cipher is implemented using Python and has been tested using several message and key combinations. Mirror cipher has applied Shanon's diffusion and confusion property and proven to be secured from bruteforce and frequency analysis attack.

1. Introduction

1.1. Background

In this modern world, data or messages are exchanged anytime and anywhere. To protect confidentiality and integrity of messages, people usually encrypt their messages before sending them, and then decrypt the received messages before reading them. These encryption and decryption practices and techniques are contained under the big concept of cryptography. There are many ciphers (encryption and decryption algorithms) that have been developed since the BC period. Ciphers are then divided into 2 kinds of ciphers, based on how it treats the message: stream cipher and block cipher.

1.2. Analysis of Other Ciphers

Block cipher is a type of cipher that operates on a fixed-length group of bits, called block. A message would be splitted into blocks and then encrypted. The Data Encryption Standard (DES) specifies two FIPS approved cryptographic algorithms which are used for encrypting (enciphering) and decrypting (deciphering) binary coded information [4]. DES splits message into 64 bits blocks, and runs 16 rounds of round function. Its 56 bits round keys are generated from the user's key. The AES (Advanced Encryption Standard) is another standard issued by NIST. Its algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits [5]. AES runs 10 rounds of round function which consists of 4 functions: subbytes, shiftrows, mixcolumns, and addroundkey.

1.3. Approach to Develop Mirror Cipher

Mirror cipher is a cipher which developed out of creativity and admiration for the nature of symmetry. It is called a mirror cipher because it has a specific feature of mirrored round function utilities. Mirror

cipher runs 9 rounds of feistel networks. This cipher is an ECB cipher which encrypts and decrypts each message block independently.

2. Theories

There are several theories which are referred to as basic knowledge in the development of mirror cipher. The theories are about cryptography in general, ciphers, block ciphers, feistel network, diffusion, and confusion.

2.1. Cryptography in General

Cryptography is the practice and study of technique for secure communication in the presence of third parties called adversaries. In general, cryptography is divided into classic cryptography and modern cryptography. Modern cryptography is based on mathematical theory and computer science practice. Classic cryptography focuses on message confidentiality. On the other hand, modern cryptography focuses on confidentiality, integrity, and non-repudiation.

2.2. Ciphers

Cipher is an algorithm to perform encryption and decryption. Encryption is a conversion of plaintext into ciphertext. Decryption is a conversion of ciphertext into plaintext. Ordinary information or comprehensible messages are called plaintext. Unintelligible information or incomprehensible messages are called ciphertext. Cipher is usually implemented as either block cipher or stream cipher. Cipher uses one or two keys as information to do encryption and decryption. Based on how key used, cipher is divided into:

- Symmetric-key cipher
  A key is used to do both encryption and decryption. To keep the information secure, key should be treated as secret information.
- Asymmetric-key cipher
  There are two types of key used in this cipher, those are public key and private key. Public key is a key that can be disseminated widely and openly. Contrarily, private key should be kept secure by the owner. The key for encryption and decryption process should be different.

2.3. Block Cipher

Block cipher is a type of cipher that operates on a fixed-length group of bits, called block. To work on long plaintext, plaintext should be divided into blocks with a predetermined size, then encrypt those blocks using one of these modes:

- Electronic Code Book (ECB)
  In this mode, each plaintext block (Pi) is individually encrypted into a ciphertext block (Ci). For decryption, each ciphertext block (Ci) is individually decrypted into a plaintext block (Pi). Figure 1 shows how ECB do encryption and figure 2 shows how ECB do decryption. The encryption formula

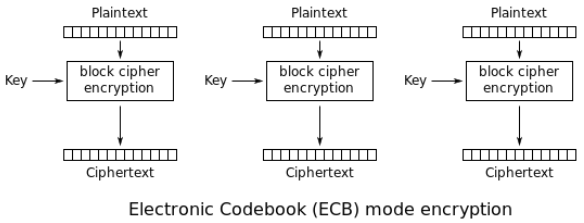

Figure 1. ECB illustration on encryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB

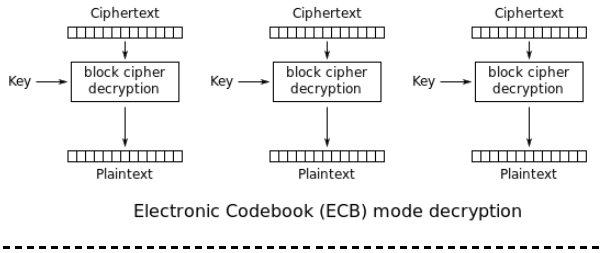

Figure 2. ECB illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB

- Cipher Block Chaining (CBC)
  For encryption, each ciphertext (Ci) block relies on all of the previous ciphertext blocks (C1, C2, …, Ci-1). The previous ciphertext block is used to xor with the next plaintext block before encryption. To make each message unique, an initialization vector (IV) is used on first block encryption. Initialization vector is a random value that has the same size as a block. Figure 3 shows how CBC do encryption and figure 4 shows how CBC do decryption.

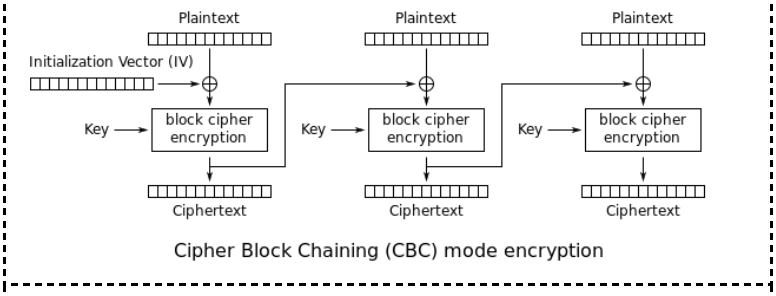

Figure 3. CBC illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB

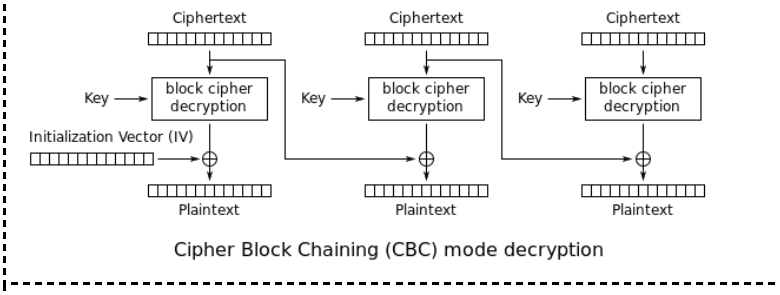

Figure 4. CBC illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#ECB

- Cipher Feedback (CFB)
  Similar to CBC, in encryption, CFB uses encrypted previous ciphertext (Ci-1) to xor with next plaintext (Pi). Initialization vector is used for encrypting and decrypting the first block. Figure 5 shows how CFB do encryption and figure 6 shows how CFB do decryption.

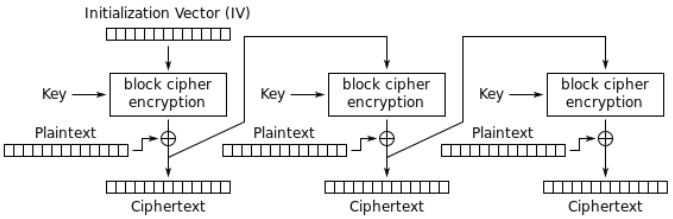

Figure 5. CFB illustration on encryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Feedback_(CFB)

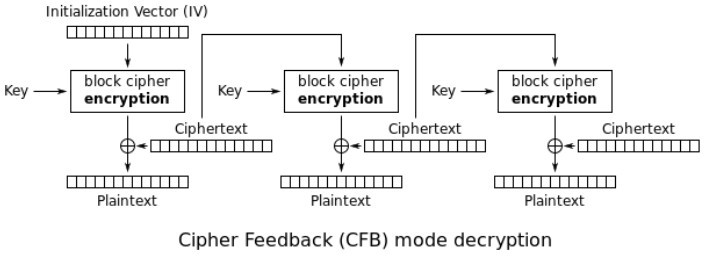

Figure 6. CFB illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Feedback_(CFB)

- Output Feedback (OFB)
  Similar to CFB, in encryption and decryption, OFB apply xor operation to keystream block with plaintext or ciphertext. Keystream block is a product of repeatedly doing encryption on initialization vector. Figure 7 shows how OFB do encryption and figure 8 shows how OFB do decryption.

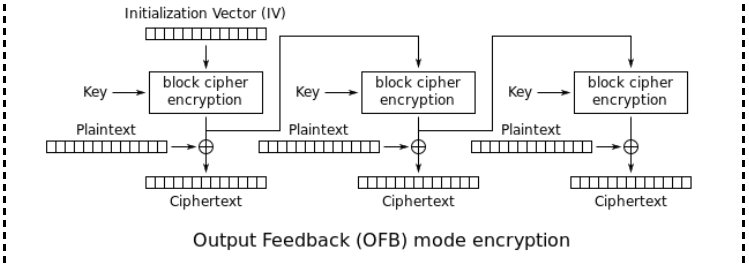

Figure 7. OFB illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Output_Feedback_(OFB)

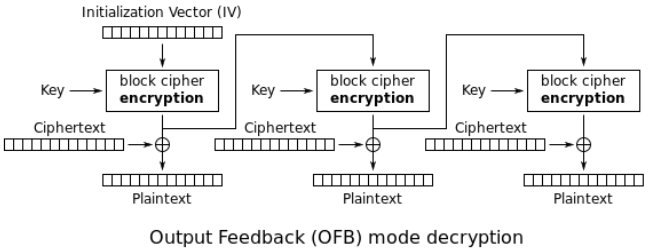

Figure 8. OFB illustration on decryption, source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Output_Feedback_(OFB)

- Counter Mode (CTR)
  In encryption and decryption, CTR do xor operation between plaintext or ciphertext with a keystream. Keystream is the result of encrypting the counter of the current block by using a key.

2.4. Feistel Network

Feistel network is a cryptographic technique used in construction of block cipher. The feistel network has reversible property. This property makes one algorithm to be used on encrypting and decrypting. The algorithm used on the feistel network is independent from the model itself. Figure 9 illustrates a simple feistel network on encryption. The F denotes an algorithm used on the feistel network. In encryption and decryption, the feistel network divides the block into two parts, left (L) and right (R). After some operation on those parts, the position of the left and the right part are swapped and concatenated into a block.

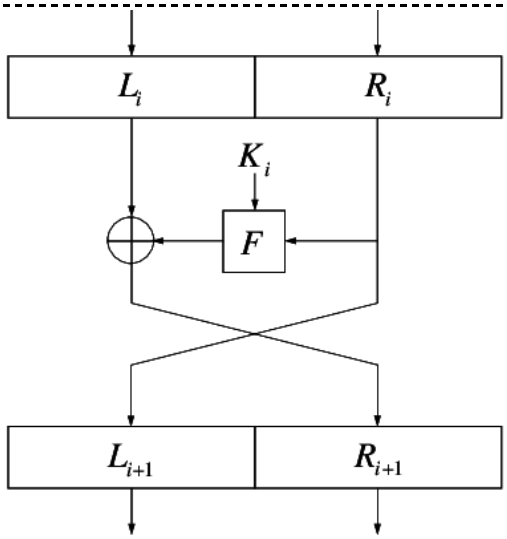

Figure 9. Encryption in feistel network , source: https://www.researchgate.net/figure/Illustration-of-a-round-in-a-Feistel-network_fig1_224645711

2.5. Confusion and Diffusion

Confusion and diffusion are two properties of secure cipher. they were identified by Claude Shanon. Confusion means that each binary digit or bit of the ciphertext should depend on several parts of the

key. Diffusion means that if a single bit of plaintext or key is changed, then half of the bits in the ciphertext should change. These properties obscures the relationship between ciphertext, plaintext, and the key.

3. Proposed Cipher Design

Mirror cipher is a feistel cipher with mirrored round function design. It uses a feistel network with round function that consist of 9 steps: s-box substitution, row substitution, column substitution, column cumulative xor, round key addition, column cumulative xor, column substitution, row substitution, and s-box2 substitution. Four earliest steps are the same as the four latest ones. That is why the cipher is called a mirror cipher.

3.1. Initial Processing

This cipher works for 256 bits message. If the message is longer than 256 bits, the message would be divided to blocks of 256 bits. If the message is shorter than 256 bits, it would be padded with a 1 and 0s until at the end of message it has 256 bits. The decryption algorithm would be the same as the encryption algorithm only with the reversed feistel network. Mirror cipher accepts key input from its users. The length of the key would be used as the seed of random function to create round key constants. The first 128 bits of the key would be used as an initialization vector. If the key is less than 128 bits, the key would be repeated until it reaches 128 bits long.

3.2. Feistel Network Design

The design of the feistel network in mirror cipher uses the original feistel network design as shown in Figure 9. The 256 bits message block would be splitted into L and R 128 bits blocks. Then, the R block would be inserted into the round function with a specific round key. The first round key would be 128 bits initialization vector which is extracted from the user input key. The next round key would be generated automatically using the previous round's key and a random seed, which is the length of the user input key. The result from the round function would be xored with the L block and then used as the new R block. The new L block would be a copy of the old R block. This design would be run 9 times in mirror cipher. The key, initialization vector, and seed would be the same for all blocks (not linked among blocks), which means mirror cipher is an ECB block cipher.

3.3. Round Function Design

The round function consists of 5 different functions: s-box substitution, row substitution, column substitution, column cumulative xor, and round key addition. Four functions are done twice. The 128 bits message is converted to hexadecimal message and being visualized as 4x4 matrix which each cell consists 2 hexadecimals.

45	4e	4b	49
52	50	53	49
20	44	45	4b
52	49	50	53

Figure 10. Message example

3.2.1. S-Box Substitution

In this round function, s-box substitution would be done twice with different s-boxes, which is in the first and last step. S-box is simply a 16x16 matrix which could be used to search substitution of each cell on our message matrix. For example on encryption using Figure 11. s-box, we want to look for a substitution for 45 (first cell on our matrix). To do that, we should look for the intersection between the fourth (first hexadecimal in the cell) row and fifth (second hexadecimal in the cell) column, which would give us 6e.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 11. First S-Box, source: https://captanu.wordpress.com/2015/04/25/aes/

Doing this to all cells in the message matrix, we would get another message matrix as shown in Figure 12. In the last step, we would use another s-box which is an inversed version of s-box used in the encryption phase.

6e	2f	b3	3b
00	53	ed	3b
b7	1b	6e	b3
00	3b	53	ed

Figure 12. Message after s-box substitution

3.2.2. Row Substitution

Row substitution is done twice in this function. It is done as the second step and eighth step. In the second step, the cipher substitutes the first row of message matrix with the third row, and the second row with the fourth row. In the eighth step, the cipher substitutes the first row of message matrix with the fourth row, and the second row with the third row. Figure 13. is showing us the result of row substitution (in second step) at the message matrix from Figure 12.

b7	1b	6e	b3
00	3b	53	ed
6e	2f	b3	3b
00	53	ed	3b

Figure 13. Message after row substitution in second step

3.2.3. Column Substitution

In this round function, column substitution is done twice. It is done as the third step and seventh step. In the third step, the cipher substitutes the first column of message matrix with the fourth column, and the second column with the second column. In the seventh step, the cipher substitutes the first column of message matrix with the third column, and the second row with the fourth row. For example, we apply column substitution (third step) to the message matrix we have in Figure 13., the result would look like Figure 14.

b3	6e	1b	b7
ed	53	3b	00
3b	b3	2f	6e
3b	ed	53	00

Figure 14. Message after column substitution in third step

3.2.4. Column Cumulative XOR

Column cumulative XOR is, as the name says, a function that does xor cumulatively to message matrix's columns. This function is done twice, as fourth and sixth step. The function does XOR the original first and second column and uses the result as the new second column. Then, it does XOR on that new second column with the original third column, and uses the result as the new third column. Lastly, it takes the new third column and XOR it with the original fourth column to get the new fourth column. Figure 15. shows the result of column cumulative xor of message matrix in Figure 14.

b3	dd	c6	71
ed	be	85	85
3b	88	a7	c9
3b	d6	85	85

Figure 15. Message after column cumulative XOR

3.2.5. Round Key Addition

Round key addition is the fifth step of this round key function. This step adds a specific generated round key to the message using xor function. The first column of the message matrix is xored with the first column of the key matrix, and so on. For example, the round key is shown at Figure 16. and the cipher adds it to the message in Figure 15.. The result is shown at Figure 17.

49	46	34	30
32	30	20	41
44	41	4c	41
48	20	4b	55

Figure 16. Round key example

fa	9b	f2	41
df	8e	a5	c4
7f	c9	eb	88
73	f6	ce	d0

Figure 17. Message after round key addition

3.2.5. Overall Round Function

The overall round function would consist of 9 steps which look like this pseudocode:

```
round_function:
    s-box substitution with s-box in Figure 11.
    row substitution:
        substitute first row and third row
        substitute second row and fourth row
    column substitution:
        substitute first column and fourth column
        substitute second column and third column
    column cumulative xor:
        new_first_column = old first column
        new_second_column = xor first column with second column
        new_third_column = xor new second column with third
column
        new_fourth_column = xor new third column with fourth
column
    add round key:
        xor with round key
    column cumulative xor
    column substitution:
        substitute first column and third column
        substitute second column and fourth column
    row substitution:
        substitute first row and fourth row
        substitute second row and third row
    s-box substitution with the inverse of s-box in Figure 11.
```

The Figure 18. also shows the process of a mirror cipher's round function.

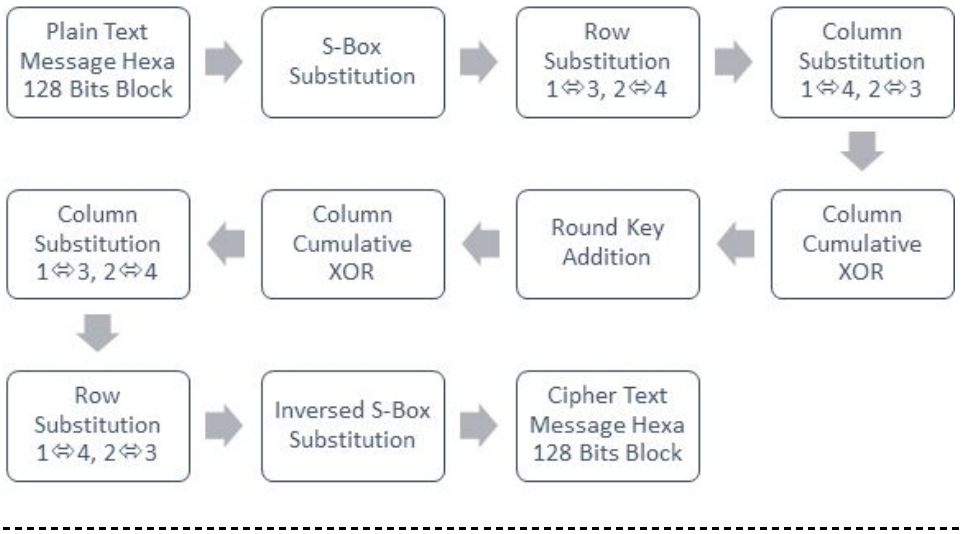

Figure 18. Diagram of the process in round function

3.4. Round Key Generation

The round key generation is a process to generate a round key for each round in the feistel network. The first round (round 0) uses 128 bits initialization vector which is extracted from the user input key. From the second round, the round key is generated from the previous round's key. For example, the second round key is generated by shifting all columns in the first round key (initialization vector) to the left, so the second column becomes the first, the third one becomes the second, and so on. After that, we take the first and last column, xor them, and use the result as the new last column. Lastly, we xor that new key matrix with round key constant by columns. This round key constant is created randomly using seed that is inputted by the user. For example, if we have an initialization vector (previous round's key) as shown in Figure 16., we shift all the columns to the left once, so it would be a new matrix as shown in Figure 19. After that, we xor the first and last column, and use the result as the last column. The matrix would now as shown in Figure 20.

46	34	30	49
30	20	41	32
41	4c	41	44
20	4b	55	48

Figure 19. Shifted previous round's key

46	34	30	f0
30	20	41	20
41	4c	41	56
20	4b	55	80

Figure 20. The round key matrix after xor first and last column

Lastly, we xor the matrix with round constant in Figure 21, so we get the new round key as shown in Figure 22.

18	2d	83	1c
aa	15	4a	a7
50	d2	c2	0a
e3	12	44	09

Figure 21. Round constant for round 1 with seed 123

5e	19	b3	ec
9a	35	0b	87
11	9e	83	5c
c3	59	11	89

Figure 22. The round key matrix after xor with round constant

4. Simulation and Result Discussion

4.1. Implementation

The implementation is using Python as coding language. The program accepts message, key, and seed from the user in MirrorCipher class. After that, the program splits the message into 256 bits blocks, and runs 9 rounds of feistel network for every block by calling FeistelNetwork class. This class takes the right 128 bits from the block and calls FeistelFunction class which contains the round function of the cipher.

This round function calls 5 different functions: `look_up_sbox` for s-box substitution, `row_substitution` for row substitution, `col_substitution` for column substitution, `cumulative_xor_col` for column cumulative xor, and `add_round_key` for round key addition.

In s-box substitution, the s-box is extracted from a file, which is read by the `read_matrix` function. In round key addition, the program uses `get_key_constant` function to get a key constant for a specific round. The key generation is implemented in the next_round_key function. This function and round key addition function use 2 helper functions: `get_column` to get a specific column, and `arrange_col` for arranging columns into a complete matrix.

There are also some helper classes: `matrix_to_string` to convert matrix to string, `string_to_matrix` to convert string to matrix, and `split_string_n_lengths` to split a string into n long strings.

4.2. Testing and Evaluation

To evaluate and test the cipher, we use this following example of user input:

```
text = ENKRIPSI DAN DEKRIPSI ADALAH PROSES YANG UTAMA DALAM KRIPTOGRAFI
key = NANIN TONY IMBA
```

The text is then converted into hexadecimal:

```
454e4b52495053492044414e2044454b52495053492041444144414c41482050524f534
5532059414e47205554414d412044414c414d204b524950544f4752414649
```

which is then splitted into 256 bits blocks:

```
['454e4b52495053492044414e2044454b52495053492041444144414c41482050524f'
,
'5345532059414e47205554414d412044414c414d204b524950544f4752414649']
```

The blocks are encrypted using a feistel network for 9 rounds. The encrypted hexadecimal result would be:

95efdd28806e41af7b9eb0afd00791f4c8cff79dce3d455e282a0f2b02e801f51e2d0d467d25a2620f8002ebaf1e362f76207ab1865caf0918f60ebeb9385965

The whole encryption process could be seen in the log in Figure 23:

```
INFO:root:Encrypting 4869647570206c65626968206461726920686172757320736 56b656461722076
INFO:root:Blocks used: 20686172757320736 56b656461722076
DEBUG:root:S-Box: [['63', '7c', '77', '7b', 'f2', '6b', '6f', 'c5', '30', '01', '67', '2b', 'fe', 'd7', 'ab', '76'], ['ca', '82', 'c9', '7d', 'fa',
'59', '47', 'f0', 'ad', 'd4', 'a2', 'af', '9c', 'a4', '72', 'c0'], ['b7', 'fd', '93', '26', '36', '3f', 'f7', 'cc', '34', 'a5', 'e5', 'f1', '71',
'd8', '31', '15'], ['04', 'c7', '23', 'c3', '18', '96', '05', '9a', '07', '12', '80', 'e2', 'eb', '27', 'b2', '75'], ['09', '83', '2c', '1a', '1b',
'6e', '5a', 'a0', '52', '3b', 'd6', 'b3', '29', 'e3', '2f', '84'], ['53', 'd1', '00', 'ed', '20', 'fc', 'b1', '5b', '6a', 'cb', 'be', '39', '4a',
'4c', '58', 'cf'], ['d0', 'ef', 'aa', 'fb', '43', '4d', '33', '85', '45', 'f9', '02', '7f', '50', '3c', '9f', 'a8'], ['51', 'a3', '40', '8f', '92',
'9d', '38', 'f5', 'bc', 'b6', 'da', '21', '10', 'ff', 'f3', 'd2'], ['cd', '0c', '13', 'ec', '5f', '97', '44', '17', 'c4', 'a7', '7e', '3d', '64',
'5d', '19', '73'], ['60', '81', '4f', 'dc', '22', '2a', '90', '88', '46', 'ee', 'b8', '14', 'de', '5e', '0b', 'db'], ['e0', '32', '3a', '0a', '49',
'06', '24', '5c', 'c2', 'd3', 'ac', '62', '91', '95', 'e4', '79'], ['e7', 'c8', '37', '6d', '8d', 'd5', '4e', 'a9', '6c', '56', 'f4', 'ea', '65',
'7a', 'ae', '08'], ['ba', '78', '25', '2e', '1c', 'a6', 'b4', 'c6', 'e8', 'dd', '74', '1f', '4b', 'bd', '8b', '8a'], ['70', '3e', 'b5', '66', '48',
'03', 'f6', '0e', '61', '35', '57', 'b9', '86', 'c1', '1d', '9e'], ['e1', 'f8', '98', '11', '69', 'd9', '8e', '94', '9b', '1e', '87', 'e9', 'ce',
'55', '28', 'df'], ['8c', 'a1', '89', '0d', 'bf', 'e6', '42', '68', '41', '99', '2d', '0f', 'b0', '54', 'bb', '16']]
DEBUG:root:Substituted by first S-Box: b745ef409d8fb78f4d7f4d43ef40b738
DEBUG:root:After row transform: ['4d7f4d43ef40b738b745ef409d8fb78f']
DEBUG:root:After col transform: ['434d7f4d38b740ef40ef45b78fb78f9d']
DEBUG:root:After col xor: 430e713c388fcf2040afea5d8f38b72a
INFO:root:Key used: 4b524950544f4752414649204153494b
DEBUG:root:Add key: 4b524950544f4752414649204153494b with 430e713c388fcf2040afea5d8f38b72a
INFO:root:After addition with round key: 086c01ce5cc0e96b3888a3fe6c727d61
DEBUG:root:After col xor: ['086465ab5c9c751e38b013ed6c1e6302']
DEBUG:root:After col transform: ['65ab0864751e5c9c13ed38b063026c1e']
DEBUG:root:After row transform: ['63026c1e13ed38b0751e5c9c65ab0864']
DEBUG:root:Inverse S-Box: [['52', '09', '6a', 'd5', '30', '36', 'a5', '38', 'bf', '40', 'a3', '9e', '81', 'f3', 'd7', 'fb'], ['7c', 'e3', '39',
'82', '9b', '2f', 'ff', '87', '34', '8e', '43', '44', 'c4', 'de', 'e9', 'cb'], ['54', '7b', '94', '32', 'a6', 'c2', '23', '3d', 'ee', '4c', '95',
'0b', '42', 'fa', 'c3', '4e'], ['08', '2e', 'a1', '66', '28', 'd9', '24', 'b2', '76', '5b', 'a2', '49', '6d', '8b', 'd1', '25'], ['72', 'f8', 'f6',
'64', '86', '68', '98', '16', 'd4', 'a4', '5c', 'cc', '5d', '65', 'b6', '92'], ['6c', '70', '48', '50', 'fd', 'ed', 'b9', 'da', '5e', '15', '46',
'57', 'a7', '8d', '9d', '84'], ['90', 'd8', 'ab', '00', '8c', 'bc', 'd3', '0a', 'f7', 'e4', '58', '05', 'b8', 'b3', '45', '06'], ['d0', '2c', '1e',
'8f', 'ca', '3f', '0f', '02', 'c1', 'af', 'bd', '03', '01', '13', '8a', '6b'], ['3a', '91', '11', '41', '4f', '67', 'dc', 'ea', '97', 'f2', 'cf',
'ce', 'f0', 'b4', 'e6', '73'], ['96', 'ac', '74', '22', 'e7', 'ad', '35', '85', 'e2', 'f9', '37', 'e8', '1c', '75', 'df', '6e'], ['47', 'f1', '1a',
'71', '1d', '29', 'c5', '89', '6f', 'b7', '62', '0e', 'aa', '18', 'be', '1b'], ['fc', '56', '3e', '4b', 'c6', 'd2', '79', '20', '9a', 'db', 'c0',
'fe', '78', 'cd', '5a', 'f4'], ['1f', 'dd', 'a8', '33', '88', '07', 'c7', '31', 'b1', '12', '10', '59', '27', '80', 'ec', '5f'], ['60', '51', '7f',
'a9', '19', 'b5', '4a', '0d', '2d', 'e5', '7a', '9f', '93', 'c9', '9c', 'ef'], ['a0', 'e0', '3b', '4d', 'ae', '2a', 'f5', 'b0', 'c8', 'eb', 'bb',
'3c', '83', '53', '99', '61'], ['17', '2b', '04', '7e', 'ba', '77', 'd6', '26', 'e1', '69', '14', '63', '55', '21', '0c', '7d']]
INFO:root:Substituted by invers first S-Box: 006ab8e9825376fc3fe9a71cbc0ebf8c
INFO:root:XOR 4869647570206c65626968206461726 9 with 006ab8e9825376fc3fe9a71cbc0ebf8c
```

Figure 23. Screenshot of program encryption log

After that, we would try to decrypt the ciphertext back to the original message. We input the ciphertext and the same key used in encryption. The program would convert the ciphertext into hexadecimals, split it into blocks, and run the inverse feistel network on that blocks using the inputted key. The program decryption log is shown in Figure …

```
INFO:root:Blocks used: f1fe4643290e3283b3293fd8923cfc2e
DEBUG:root:S-Box: [['63', '7c', '77', '7b', 'f2', '6b', '6f', 'c5', '30', '01', '67', '2b', 'fe', 'd7', 'ab', '76'], ['ca', '82', 'c9', '7d', 'fa',
'59', '47', 'f0', 'ad', 'd4', 'a2', 'af', '9c', 'a4', '72', 'c0'], ['b7', 'fd', '93', '26', '36', '3f', 'f7', 'cc', '34', 'a5', 'e5', 'f1', '71',
'd8', '31', '15'], ['04', 'c7', '23', 'c3', '18', '96', '05', '9a', '07', '12', '80', 'e2', 'eb', '27', 'b2', '75'], ['09', '83', '2c', '1a', '1b',
'6e', '5a', 'a0', '52', '3b', 'd6', 'b3', '29', 'e3', '2f', '84'], ['53', 'd1', '00', 'ed', '20', 'fc', 'b1', '5b', '6a', 'cb', 'be', '39', '4a',
'4c', '58', 'cf'], ['d0', 'ef', 'aa', 'fb', '43', '4d', '33', '85', '45', 'f9', '02', '7f', '50', '3c', '9f', 'a8'], ['51', 'a3', '40', '8f', '92',
'9d', '38', 'f5', 'bc', 'b6', 'da', '21', '10', 'ff', 'f3', 'd2'], ['cd', '0c', '13', 'ec', '5f', '97', '44', '17', 'c4', 'a7', '7e', '3d', '64',
'5d', '19', '73'], ['60', '81', '4f', 'dc', '22', '2a', '90', '88', '46', 'ee', 'b8', '14', 'de', '5e', '0b', 'db'], ['e0', '32', '3a', '0a', '49',
'06', '24', '5c', 'c2', 'd3', 'ac', '62', '91', '95', 'e4', '79'], ['e7', 'c8', '37', '6d', '8d', 'd5', '4e', 'a9', '6c', '56', 'f4', 'ea', '65',
'7a', 'ae', '08'], ['ba', '78', '25', '2e', '1c', 'a6', 'b4', 'c6', 'e8', 'dd', '74', '1f', '4b', 'bd', '8b', '8a'], ['70', '3e', 'b5', '66', '48',
'03', 'f6', '0e', '61', '35', '57', 'b9', '86', 'c1', '1d', '9e'], ['e1', 'f8', '98', '11', '69', 'd9', '8e', '94', '9b', '1e', '87', 'e9', 'ce',
'55', '28', 'df'], ['8c', 'a1', '89', '0d', 'bf', 'e6', '42', '68', '41', '99', '2d', '0f', 'b0', '54', 'bb', '16']]
DEBUG:root:Substituted by first S-Box: a1bb5a1aa5ab23ec6da575614febb031
DEBUG:root:After row transform: ['6da575614febb031a1bb5a1aa5ab23ec']
DEBUG:root:After col transform: ['6175a56d31b0eb4f1a5abba1ec23aba5']
DEBUG:root:After col xor: 6114b1dc31816a251a40fb5aeccf64c1
INFO:root:Key used: 4b524950544f4752414649204153494b
DEBUG:root:Add key: 4b524950544f4752414649204153494b with 6114b1dc31816a251a40fb5aeccf64c1
INFO:root:After addition with round key: 2a655bad46ce069cf82db22d8c777a8a
DEBUG:root:After col xor: ['2a4f14b946888e12f8d5674a8cfb810b']
DEBUG:root:After col transform: ['14b92a4f8e124688674af8d5810b8cfb']
DEBUG:root:After row transform: ['810b8cfb674af8d58e12468814b92a4f']
DEBUG:root:Inverse S-Box: [['52', '09', '6a', 'd5', '30', '36', 'a5', '38', 'bf', '40', 'a3', '9e', '81', 'f3', 'd7', 'fb'], ['7c', 'e3', '39',
'82', '9b', '2f', 'ff', '87', '34', '8e', '43', '44', 'c4', 'de', 'e9', 'cb'], ['54', '7b', '94', '32', 'a6', 'c2', '23', '3d', 'ee', '4c', '95',
'0b', '42', 'fa', 'c3', '4e'], ['08', '2e', 'a1', '66', '28', 'd9', '24', 'b2', '76', '5b', 'a2', '49', '6d', '8b', 'd1', '25'], ['72', 'f8', 'f6',
'64', '86', '68', '98', '16', 'd4', 'a4', '5c', 'cc', '5d', '65', 'b6', '92'], ['6c', '70', '48', '50', 'fd', 'ed', 'b9', 'da', '5e', '15', '46',
'57', 'a7', '8d', '9d', '84'], ['90', 'd8', 'ab', '00', '8c', 'bc', 'd3', '0a', 'f7', 'e4', '58', '05', 'b8', 'b3', '45', '06'], ['d0', '2c', '1e',
'8f', 'ca', '3f', '0f', '02', 'c1', 'af', 'bd', '03', '01', '13', '8a', '6b'], ['3a', '91', '11', '41', '4f', '67', 'dc', 'ea', '97', 'f2', 'cf',
'ce', 'f0', 'b4', 'e6', '73'], ['96', 'ac', '74', '22', 'e7', 'ad', '35', '85', 'e2', 'f9', '37', 'e8', '1c', '75', 'df', '6e'], ['47', 'f1', '1a',
'71', '1d', '29', 'c5', '89', '6f', 'b7', '62', '0e', 'aa', '18', 'be', '1b'], ['fc', '56', '3e', '4b', 'c6', 'd2', '79', '20', '9a', 'db', 'c0',
'fe', '78', 'cd', '5a', 'f4'], ['1f', 'dd', 'a8', '33', '88', '07', 'c7', '31', 'b1', '12', '10', '59', '27', '80', 'ec', '5f'], ['60', '51', '7f',
'a9', '19', 'b5', '4a', '0d', '2d', 'e5', '7a', '9f', '93', 'c9', '9c', 'ef'], ['a0', 'e0', '3b', '4d', 'ae', '2a', 'f5', 'b0', 'c8', 'eb', 'bb',
'3c', '83', '53', '99', '61'], ['17', '2b', '04', '7e', 'ba', '77', 'd6', '26', 'e1', '69', '14', '63', '55', '21', '0c', '7d']]
INFO:root:Substituted by invers first S-Box: 919ef0630a5ce1b5e63998979bdb9592
INFO:root:XOR b334e53ef05c5047b9f6de242cca8b72 with 919ef0630a5ce1b5e63998979bdb9592
INFO:root:XOR result: 22aa155dfa00b1f25fcf46b3b7111ee0
INFO:root:Decrypt result: ('22aa155dfa00b1f25fcf46b3b7111ee0', 'f1fe4643290e3283b3293fd8923cfc2e')
INFO:root:Total decrypt result: ['22aa155dfa00b1f25fcf46b3b7111ee0f1fe4643290e3283b3293fd8923cfc2e']
```

Figure 24. Screenshot of program decryption log

The final result would be converted to string so we get our original message:

```
text = ENKRIPSI DAN DEKRIPSI ADALAH PROSES YANG UTAMA DALAM KRIPTOGRAFI
```

4.3. Frequency Analysis

Using a quite big message of 832 bytes (6656 bits), we could draw a frequency graph of 256 possible groups of two hexadecimals. Figure 25 shows the frequency graph in plaintext and ciphertext.

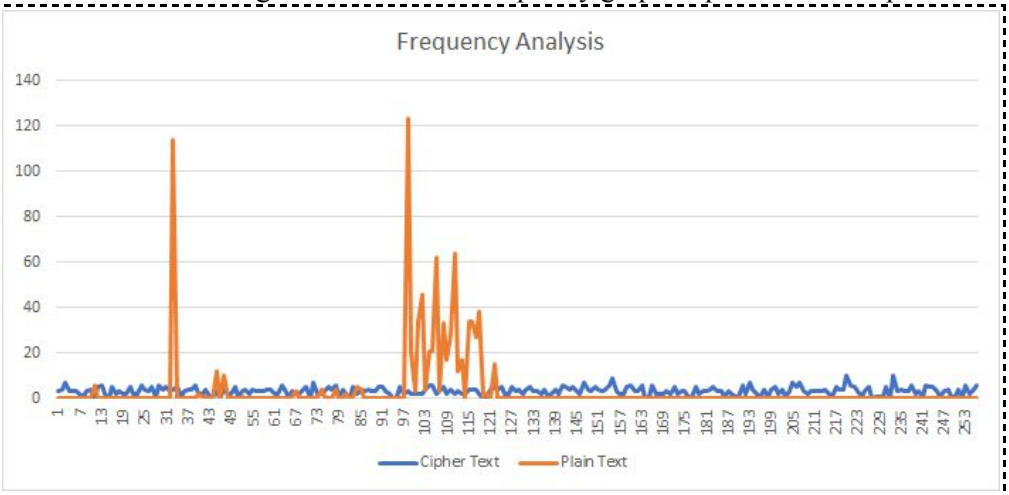

Figure 25. Frequency analysis in plaintext and ciphertext

5. Security Analysis

5.1. Bruteforce Attack Analysis

Bruteforce attack is an attack which tries all possible keys to crack the plain text message from the cipher text message. Mirror cipher uses 4 bytes or 128 bits long key. This means there are 2^{128} possible keys. If we assume that brute force attack could try 10^7 keys per second in a dual core computer, then it means we need around 3.4028×10^{31} seconds or around 1.079×10^{24} years to crack the key. Because this amount of time is quite expensive compared to the value of encrypted messages, we could conclude that mirror cipher is relatively secure from bruteforce attack.

5.2. Frequency Analysis Attack

Frequency analysis attack is an attack that compares frequency of letters in plaintext and ciphertext, and uses the similar frequencies to guess the plaintext with known ciphertext. Comparing blue and orange lines in frequency graph in Figure 25, we could conclude that even though there are some high frequency letters in plaintext (such as A, E, etc.), the ciphertext has relatively even frequency for all letters. This means that the mirror cipher has applied Shanon's confusion property.

5.3. Diffusion and Confusion Analysis

To analyze if the mirror cipher has applied Shanon's diffusion and confusion property, we try encrypting a message with 2 different keys, encrypting 2 different plaintext with the same key, and decrypting 2 different ciphertext with the same key. The differences should be as small as one bit.
Firstly, we try encrypting this message:

```
45 4e 4b 52 49 50 53 49 20 44 45 4b 52 49 50 53 49 20 44 49 20 4b
52 49 50 54 4f 47 52 41 46 49
```

with 2 different keys

- `4e 41 4e 49 4e 20 54 4f 4e 59 20 49 48 53 41 4e`
- `4e 41 4e 49 4e 20 54 4f 4e 59 20 49 48 53 41 4f`

Then, we compare the ciphertext result:

- `33 c1 82 63 e7 f1 59 c8 2a 7f d2 8a 9c 43 82 b0 de f3 5d 94 cc f8 e3 3c b9 6e ae 25 92 b5 5f 2f`
- `d1 68 82 63 4b 69 59 c8 2a 7f d2 8a 9c 43 82 b0 6b 41 5d 94 a0 31 e3 3c b9 6e ae 25 92 b5 5f 2f`

Both ciphertexts look quite different with roughly around 32 out of 256 bits changed, even though the messages are the same and the keys only have one bit differences.
Secondly, we try encrypting these plaintext messages:

- Hidup lebih dari harus sekedar t
  `48 69 64 75 70 20 6c 65 62 69 68 20 64 61 72 69 20 68 61 72 75 73 20 73 65 6b 65 64 61 72 20 74`
- Hidup lebih dari harus sekedar u
  `48 69 64 75 70 20 6c 65 62 69 68 20 64 61 72 69 20 68 61 72 75 73 20 73 65 6b 65 64 61 72 20 75`

with the same key : KRIPTOGRAFI ASIK
The ciphertext results in hexadecimals would be:

- `4f 40 3e 24 53 0f 55 85 e6 10 dc 90 cd 6d 0f dc b9 78 5c 46 11 8b 73 2d f1 ac 51 87 06 87 58 03`
- `2a ec cf 5b 27 e2 96 f0 fc 4a 0e b4 17 71 dd 5b 7d 38 18 08 2a af 09 30 11 b9 77 ab 55 8c eb 10`

We could see that both ciphertexts look very different with around 121 out of 256 bits changed, even though the keys are the same and the messages only have one bit differences.
Lastly, we try decrypting these ciphertext messages:

- `62 69 57 77 a1 c4 0b af 40 91 a6 1f 01 5c 86 6d f9 e4 54 ad 20 ef 13 ef cb 80 4e 55 94 1a f8 d6`
- `62 69 57 77 a1 c4 0b af 40 91 a6 1f 01 5c 86 6d f9 e4 54 ad 20 ef 13 ef cb 80 4e 55 94 1a f8 d7`

with the same key : KRIPTOGRAFI ASIK
We would get plaintexts:

- `48 69 64 75 70 20 6c 65 62 69 68 20 64 61 72 69 20 68 61 72 75 73 20 73 65 6b 65 64 61 72 20 76`
- `22 aa 15 5d fa 00 b1 f2 5f cf 46 b3 b7 11 1e e0 f1 fe 46 43 29 0e 32 83 b3 29 3f d8 92 3c fc 2e`

Even though the keys are the same and the ciphertexts only have one bit differences, the plaintexts are totally different with 126 out of 256 bit changed.

From those 3 test results, we could conclude that mirror cipher has applied Shanon's diffusion property because with only one bit difference, we would get totally different results.

6. Conclusion and Future Works

Mirror cipher is a new developed ECB type block cipher, which has a specific feature of mirrored round function. Mirror cipher accepts message and key inputs from its user and runs 9 rounds of feistel networks. Mirror cipher's round function consists of 5 different functions in mirrored order, so it has 9 function calls in total. Mirror cipher could be used to secure messages' confidentiality and integrity. Mirror cipher has applied Shanon's diffusion and confusion property and it relatively could not be cracked using bruteforce or frequency analysis attack. In the future, there might be researches on different modes of mirror cipher, could be in CBC mode, CFB mode, etc.

7. References

[1] Ronald L. Rivest 1990 "Cryptography". In J. Van Leeuwen (ed.). Handbook of Theoretical Computer Science (Elseveir)

[2] "Information Theory and Entropy". Model Based Inference in the Life Sciences: A Primer on Evidence (New York: Springer) p 51-82

[3] Padding Schemes for Block Ciphers. Retrieved from https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html

[4] National Institute of Standards and Technology, *Data Encryption Standard (DES)*. (U.S.: U.S. Department of Commerce)

[5] National Institute of Standards and Technology, *Advanced Encryption Standard (DES)*. (U.S.: U.S. Department of Commerce)

Acknowledgments

First and foremost, praises and thanks to God, for His blessings and kindness, so this paper could be finished. We would like to express our deep gratitude to our lecturer Dr. Ir. Rinaldi Munir, MT. that has taught us about cryptography, especially about block ciphers and feistel network. Without that basic knowledge, we must be confused when facing this task. We also want to say thank you to Mr. Rinaldi for arranging this task so we could expand our knowledge and creativity by making a new cipher. We would never forget to thank all our colleagues, students of batch 2016 Informatics Major in Bandung Institute of Technology, for all the support. Last but not least, we want to acknowledge with gratitude the support and endless love from each of our family.