ITMFR: A New Block Cipher Algorithm with Feistel Network, Round Key, and Key-Dependent Substitution and Transposition Following Shannon's Diffusion and Confusion Principle

Ignatius Timothy Manullang[1], **Fatur Rahman**[2].

[1,2] Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung (ITB), Jalan Ganesha 10, Bandung 40132
E-mail: ignatiustimothymanullang@gmail.com , 13517056@std.stei.itb.ac.id

Abstract. Currently, many block cipher algorithms are vulnerable to linear cryptanalysis and differential cryptanalysis. In order to fix those vulnerabilities, substitution and permutation which are key-dependent, theoretically, can be incorporated into the block cipher algorithm. Therefore, a new block cipher, which is named ITMFR is presented. It incorporates Feistel network, round keys, and key dependent transposition or permutation and substitution cipher for diffusion and confusion which provides additional security and complexity into the traditional block cipher algorithm.

Keywords: Block Cipher, Feistel Network, Round Key, Key-dependent, Substitution, Transposition

1. Introduction

In this era of information, computer technology is being developed rapidly. The transmission of information happens more frequently, with the advent of new information exchange medias, including, however, not limited to social medias, which results in the ever-growing necessity of data protection against illegal duplication and unauthorized use. This is in contrast to the availability and the millions of users using the internet network which is a non-secure public channel. The development of this particular computer technology has a negative impact in which secret messages can be retrieved through various attacks. This has led to the increasing reliance on information security, which can be obtained through the application of cryptography. Cryptography, can provide security properties to the information, such as confidentiality, integrity, authentication, and non-repudiation.

In Cryptography, the message goes through the process of encryption, in which the information is encoded with a key before it is sent by the sending party. The receiving party will then receive the encrypted message. There are many cryptographic algorithms. One of them is Block Cipher. Block Cipher is an encryption method that applies a deterministic algorithm along with a symmetric key to encrypt a block of text.

The main components of the Block Cipher include substitution box (S-box), permutation box (P-box), and other encryption operations with key. S-box functions as a component for substitution cipher and provides confusion, P-box functions as a component for transposition cipher and provides diffusion. The terms diffusion and confusion were identified by Claude Shannon in his 1945 classified report *A Mathematical Theory of Cryptography* [1].

Many popular Block Ciphers include S-box and P-box which are specifically designed for better security. S-box and P-box in Block Ciphers in standards such as DES [2] and AES [3] produce non-linear transformations. S-box and P-box in Block Ciphers such as Khufu [4], DSDP [5], Twofish [6] and Blowfish [7] uses key-dependent S-box and/or P-box, in which the S-box and/or P-box is created using the encryption key.

1.1. DES

DES or Data Encryption Standard, is a standard for DEA or Data Encryption Algorithm, a symmetric-key block cipher algorithm which is used to encrypt digital data. DES was developed at IBM in 1972. DES operates with 64-bit blocks, with external key length equal to its block size of 64 bits. However, only 56 bits of its external key are used, the other 8 parity bits are not used.

In DES, each plaintext block is encrypted in 16 enciphering rounds, which follows the Feistel Network (shown in figure 2). Each round uses a different internal key, which consists of 48-bits and is generated from the 64-bit external key from the user, using key scheduling (shown in figure 3). Each block undergoes an initial permutation (IP), 16 rounds of enciphering, and initial permutation inversion (IP^{-1}) (shown in figure 1).

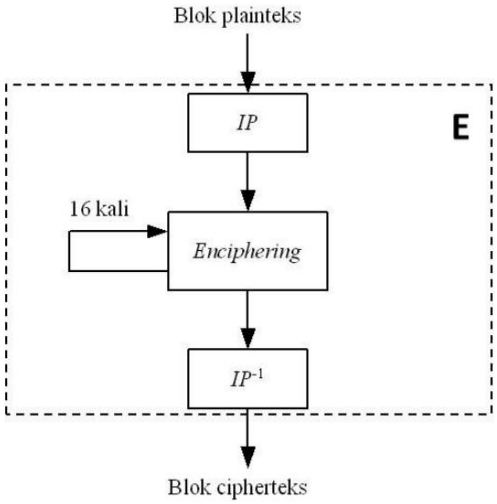


Figure 1. DES Global Scheme

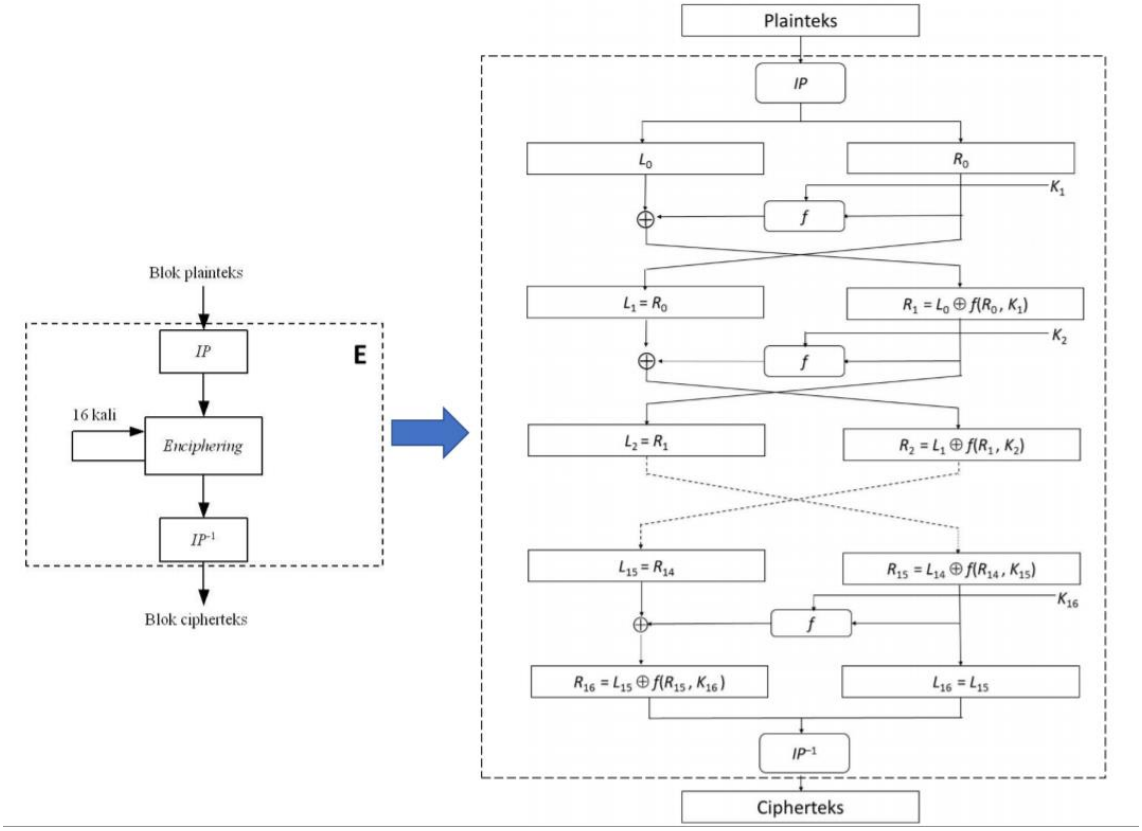


Figure 2. DES Encryption Algorithm

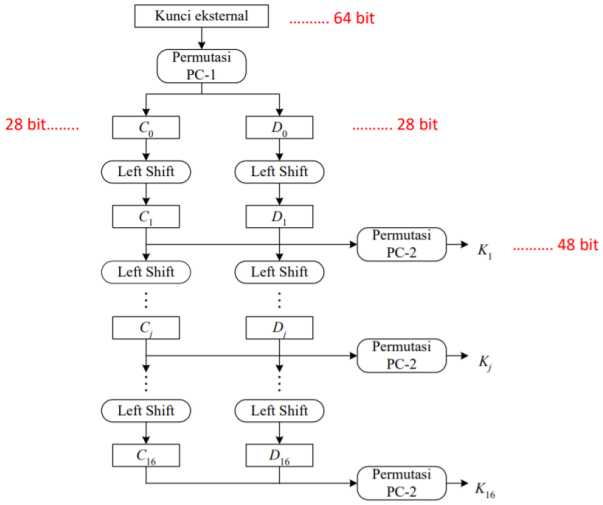


Figure 3. DES Internal key generation

The Feistel function, which is depicted in figure 2, operates on a 32-bit half-block, and consists of four stages, Expansion, Key mixing, Substitution and Permutation.

Expansion involves expanding the 32-bit half-block into 48 bits using expansion permutation by duplicating half of the bits.

Key mixing involves combining the expansion result with a subkey that corresponds to the encryption round using a XOR operation.

Substitution involves dividing the key mixing result into eight 6-bit pieces, then it is substituted using the substitution boxes or S-boxes. Each of the eight S-boxes replaces its six input bits with four output bits, following a non-linear transformation

Permutation involves rearranging the output of substitution according to the permutation box (P-box).

1.2. AES

AES, which is also known as Rijndael, is a Block Cipher standard which is designed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, and established by the U.S. National Institute of Standards and Technology (NIST).

AES is based on the substitution-permutation network. AES consists of three Block Ciphers, AES-128, AES-192 and AES-256. AES incorporates 128-bit block sizes with varying key sizes of 128 for AES-128, 192 for AES-192, 256 for AES-256, and varying number of rounds, 10 for AES-128, 12 for AES-192, 14 for AES-256. The Algorithm has some steps, which include KeyExpansion, AddRoundKey, SubBytes, ShiftRows, and MixColumns.

KeyExpansion involves deriving round keys from the cipher key using AES key schedule.

AddRoundkey involves combining each byte of the current message state with the cipher key.

SubBytes involves byte substitution using a substitution table (S-box).

ShiftRows involves shifting of rows of array state through wrapping.

MixColumns involve mixing data in each column of the state array.

The process is ordered as follows:

1. Initial Round
    1.1. AddRoundKey
2. Repeated as much as the number of rounds (Nr) - 1.
    2.1. SubBytes
    2.2. ShiftRows
    2.3. MixColumns
    2.4. AddRoundKey
3. Final round.
    3.1. SubBytes
    3.2. ShiftRows
    3.3. AddRoundKey

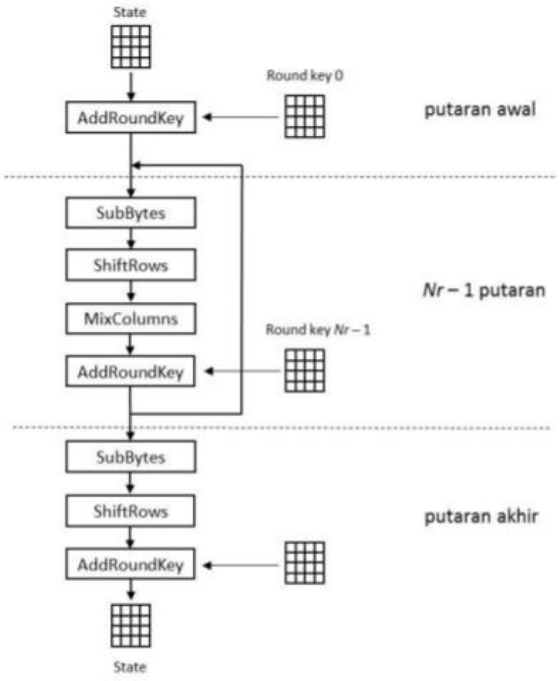


Figure 4. AES algorithm process

1.3. Khufu

Khufu is a Block Cipher which is designed by Ralph C. Merkle in 1989. The cipher is named based on the Egyptian Pharaoh Khufu. Khufu is a 64-bit block cipher that uses 512-bit keys, which is unusual considering that other block ciphers typically have much smaller keys which rarely exceed 256 bits.

Khufu incorporates the Feistel Network with 16 rounds by default, although the users are allowed to change the number of rounds into multiples of eight between 8 and 64. The S-Box used in this Block Cipher is a key-dependent S-Box which changes every 8 rounds (which is termed an *octet*).

In a round, the message is divided into two, left half and right half. The least significant bit of the left half the block is used to create the 8×32-bit S-box. Then, the S-box is XORed with the other 32-bit half. The left half is rotated to bring a new byte into position. Then, the halves are swapped. At the start and end of the algorithm, key whitening, which in this case is done by XOR between extra key material and the block.

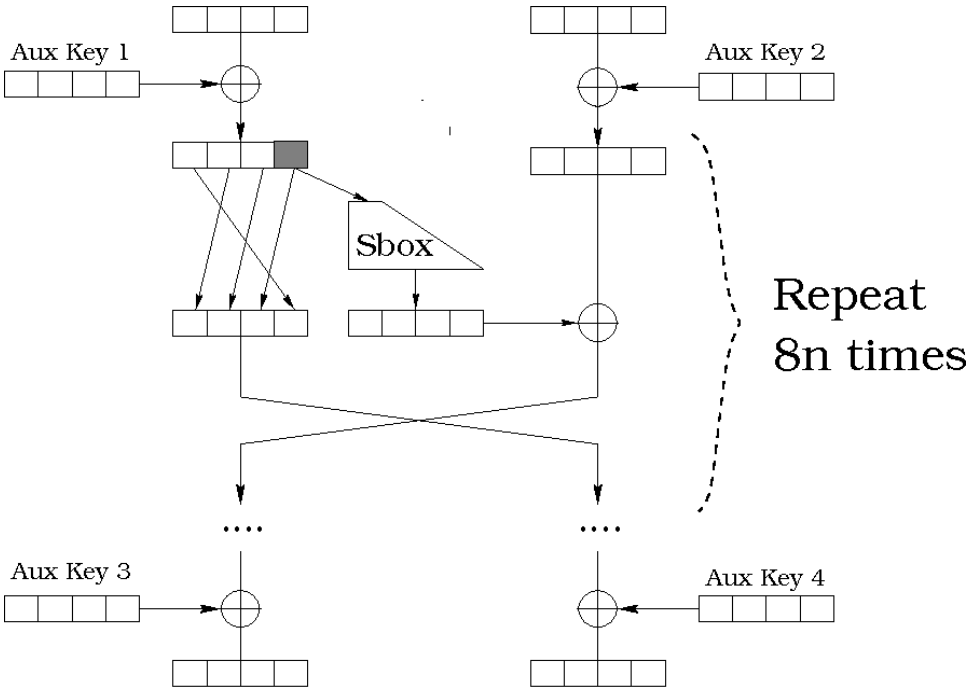


Figure 5. Khufu Structure

DSDP

DSDP is a 128-bit Block Cipher which incorporates key-dependent S-Box and P-Box. Key sizes vary but are generally more than 128 bits. The number of rounds also varies. The variation of key sizes and number of rounds makes this block cipher flexible in terms of trade-off between speed and security. Permutation and Substitution on the DSDP is done with S-Box and P-Box, which are generated by the RC4 cipher's key scheduling algorithm.

1.4. Twofish

Twofish is a symmetric key block cipher, based on feistel network, with a block size of 128 bits and key sizes up to 256 bits. The components of Twofish cipher are key-dependent S-box, Maximum-Distance Separable Matrix (MDS Matrix), Pseudo-Hadamard Transform (PHT) and whitening process. Twofish is designed to meet AES criteria (Advanced Encryption Standard) such as, the block size must be 128 bits, capable of accepting key sizes of 128 bits, 192 bits and 256 bits and efficient in both hardware and software implementation.

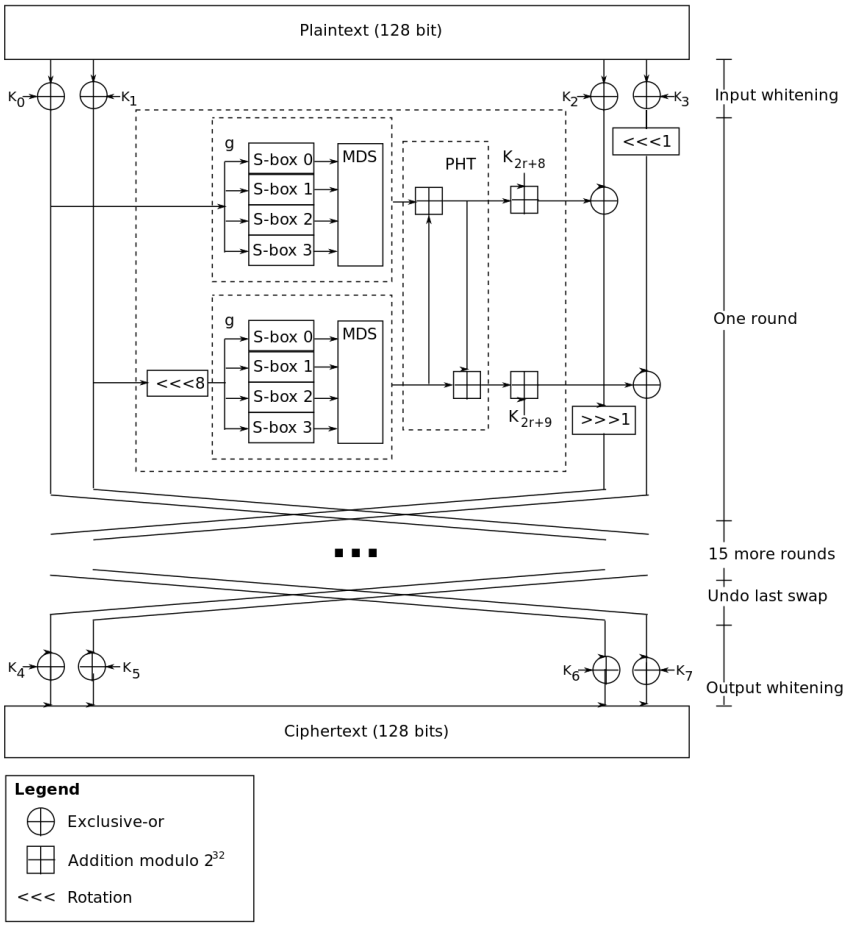


Figure 6. Twofish Structure

1.5. Blowfish

Blowfish is a symmetric key block cipher designed in 1993 by Bruce Schneier. The Blowfish algorithm incorporates a 64-bit block size and a variable key length from 32 bits up to 448 bits. It is based on the Feistel Network, in which Blowfish incorporates 16-rounds of Feistel Network. It also does substitution using large key-dependent S-boxes.

Figure 7. Twofish Structure

2. Basic Theory

2.1. Block Cipher

Block cipher [8] is a symmetric key cryptographic algorithm which operates on a block of text. In block cipher, the plaintext bits are divided into blocks of bits of equal length. Encryption on block cipher is done on plaintext blocks with bits of the key, and the user-supplied, external key does not have to equal the length of the plaintext block.

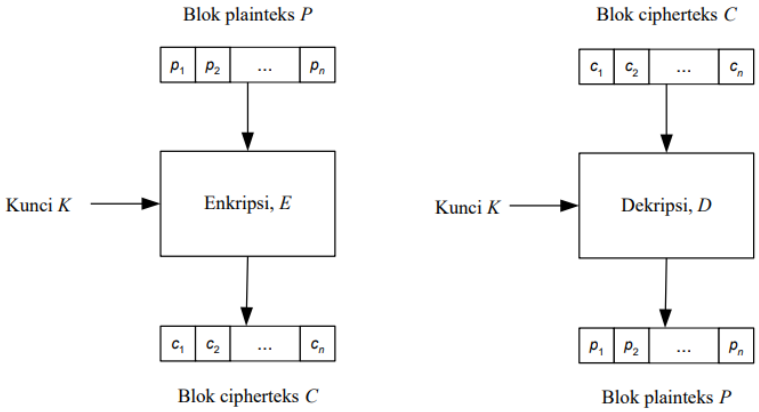


Figure 8. Block Cipher Encryption & Decryption Scheme

2.2. Block Cipher Operation Modes

Block cipher operation modes are modes in which a block is operated before it is encrypted or decrypted by the encryption or decryption function. There are 5 (five) block cipher operation modes, which are Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter Mode.

2.2.1. Electronic Code Book (ECB)

The Electronic Code Book mode or ECB mode involves individually and independently encrypting each plaintext block of P_i into a block ciphertext C_i, by using the Encryption formula:

$$C_i = E_K (P_i) \quad (1)$$

And the Decryption formula:

$$P_i = D_K (C_i) \quad (2)$$

In this case, P_i and C_i respectively are the i-th plaintext and ciphertext blocks.

In ECB mode, the same plaintext block is always encrypted to be the same block ciphertext. Since each block of the same plaintext is always encrypted into the same block of ciphertext, it is theoretically possible to create a corresponding plaintext and ciphertext codebook.

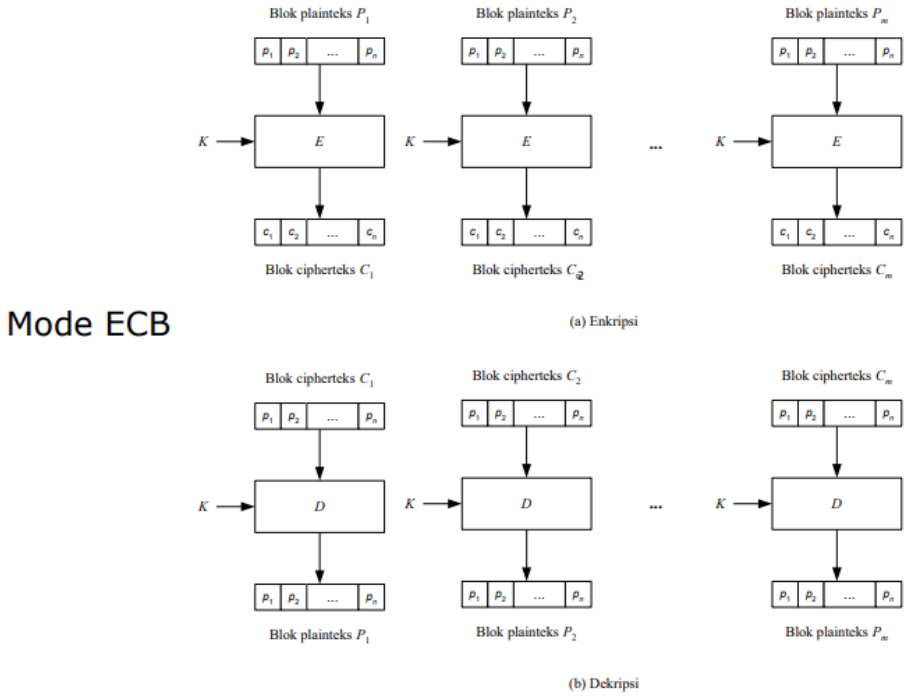


Figure 9. Encryption & Decryption Scheme of Electronic Code Book (ECB) Block Cipher Operation Mode

2.2.2. Cipher Block Chaining (CBC)

The Cipher Block Chaining mode or CBC mode aims to create dependencies between blocks. Each block of ciphertext depends on all previous plaintext blocks. The result of the previous block encryption is sent into the initialization vector (IV). The IV can be given by the user or generated randomly by the program. In the decryption process, the plaintext block is obtained by XOR-ing IV with the results of the decryption current block encryption. The first block encryption requires a pseudo block (C_0) which is called the initial of the first ciphertext block. The IV can be given by the user or generated randomly by the program. In the decryption, the plaintext block is obtained by XOR-ing IV with the results of the decryption of the first ciphertext block.

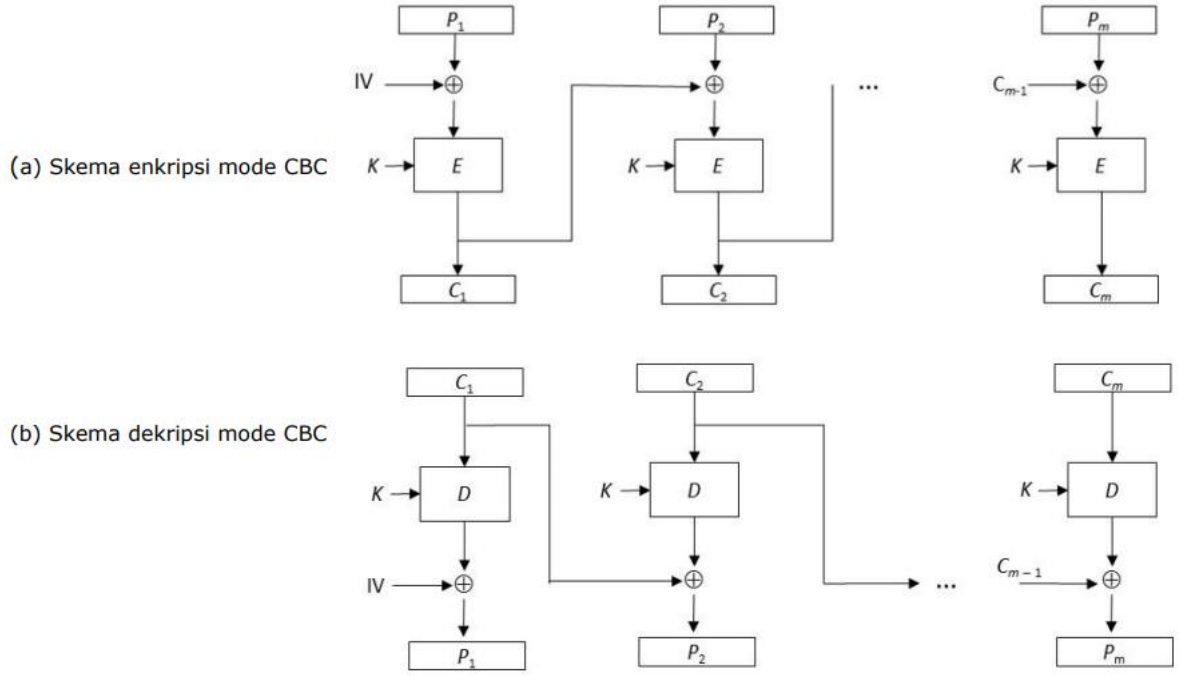
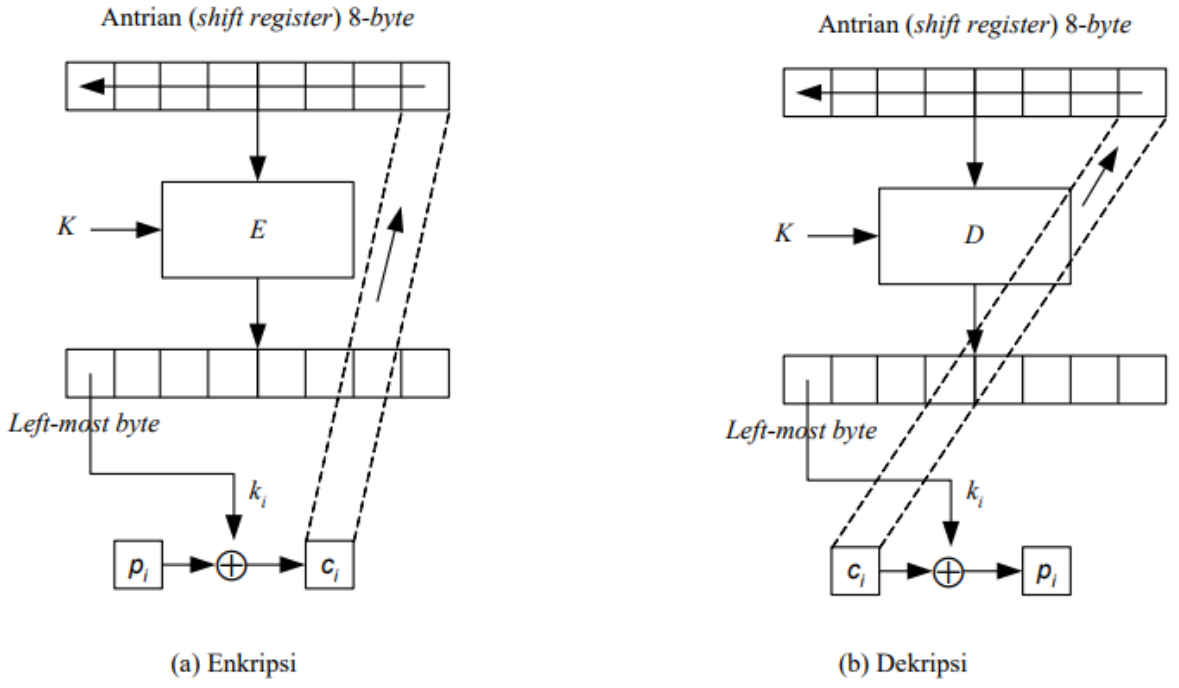


Figure 9. Encryption & Decryption Scheme of Cipher Block Chaining (CBC) Block Cipher Operation Mode

2.2.3. Cipher Feedback (CFB)

The Cipher Feedback mode or CFB mode aims to overcome weaknesses in CBC mode when applied to data communications (incomplete block size). Data is encrypted in units that are smaller than the block size. The n-bit CFB encrypts n bits of plaintext at a time, which makes the n-bit CFB treat the block ciphers similar to stream ciphers. CFB mode requires a queue that is the same size as the input block size. The 8-bit CFB mode for 64-bit (8-byte) blocks operates as follows:



(a) Enkripsi (b) Dekripsi

Figure 10. Encryption & Decryption Scheme of 8-bit CFB Cipher Feedback (CFB) Block Cipher Operation Mode for 64-bit (8-byte) blocks

2.2.4. Output Feedback (OFB)

The Output Feedback mode or OFB mode is similar to CFB mode, except that n bits of the encrypted queue are copied to the rightmost position element in the queue. Decryption is done as the reverse of the encryption process. In the encryption process, 1-bit errors in plaintext blocks only affect the corresponding ciphertext blocks, and in the decryption process, 1-bit errors in the ciphertext block only affect the corresponding plaintext blocks. The 8-bit OFB mode for 64-bit (8-byte) blocks operates as follows:

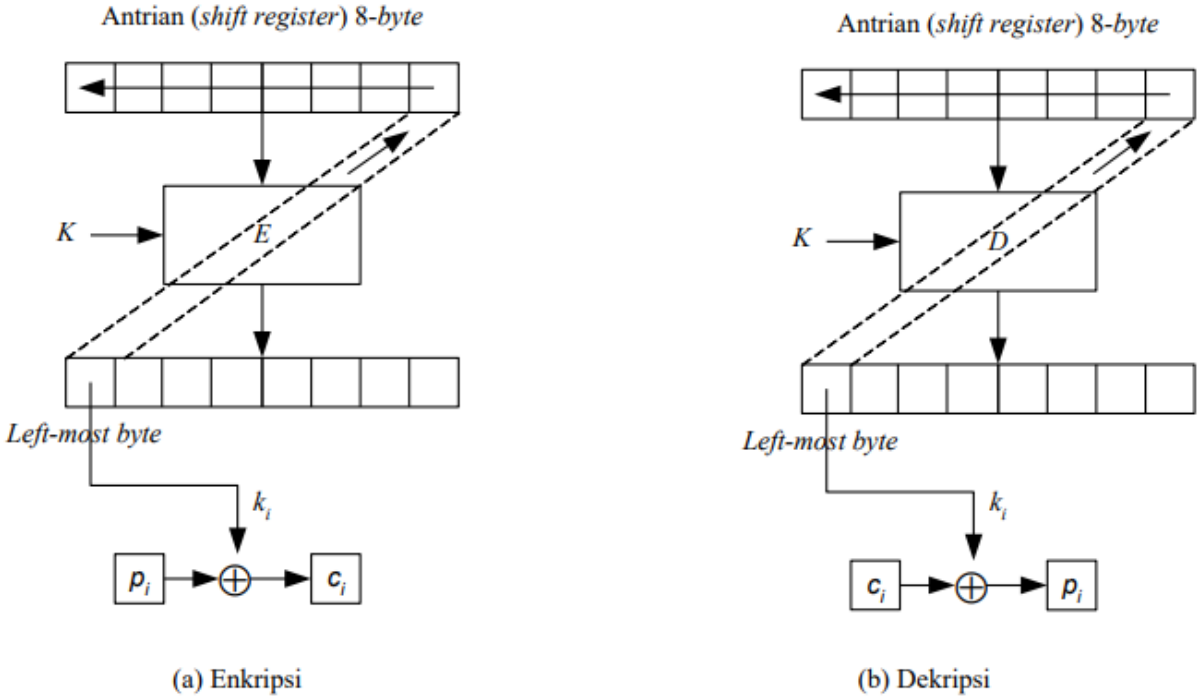


Figure 11. Encryption & Decryption Scheme of 8-bit CFB Cipher Feedback (CFB) Block Cipher Operation Mode for 64-bit (8-byte) blocks

2.2.5. Counter Mode

The Counter Mode uses counters for encryption and decryption. A counter is a value in the form of a block of bits whose size is the same as the size of the plaintext block. The counter value must be different from each encrypted block. Initially, for the first block encryption or decryption, the counter is initialized with a value. Furthermore, for the encryption or decryption of the next blocks the counter is incremented by one value. The structure of Counter Block Cipher Mode is as follows:

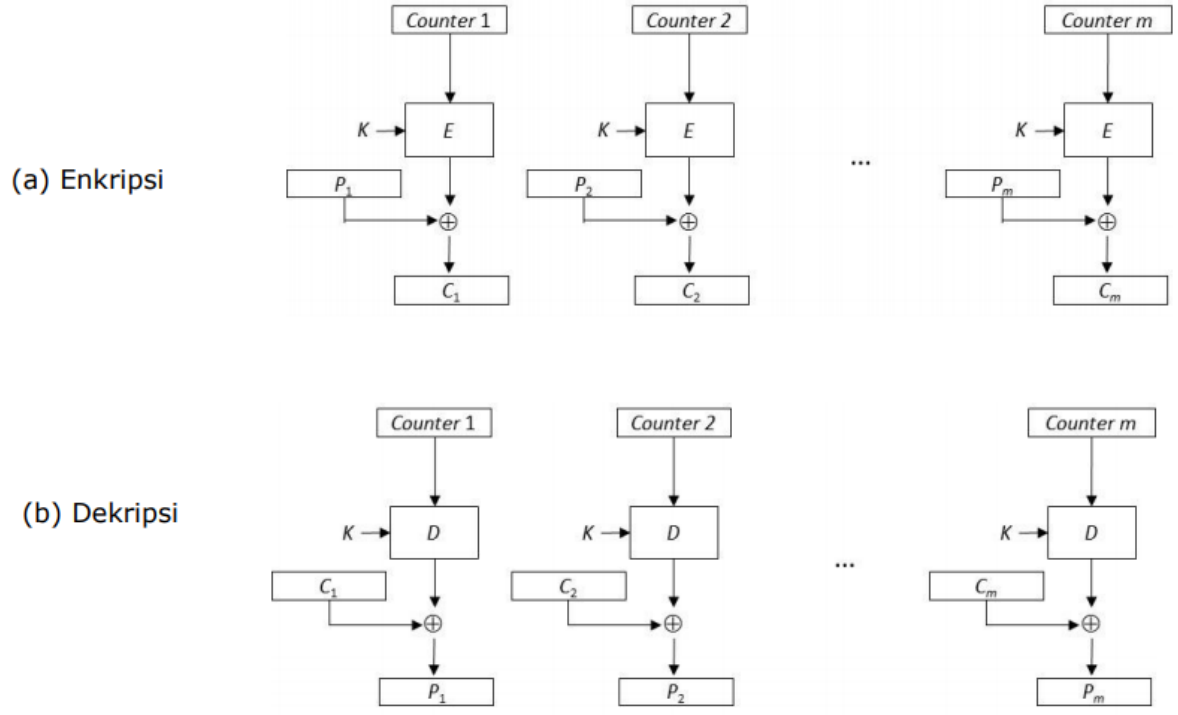


Figure 12. Encryption & Decryption Scheme of Counter Block Cipher Operation Mode

2.3. Block Cipher Structure

Block Cipher Structure is the base in which Block Ciphers are created. There are 2 types of block cipher structures, namely SPN (Substitution Permutation Network) and Feistel Network. An example of Block Cipher which is based on SPN is AES, and examples of Block Cipher which is based on the Feistel Network are DES, Twofish, Blowfish and Khufu.

SPN has 4 types of transformations, namely substitution transformations, permutation transformations, linear merge transformations and key addition transformations. The structure of SPN is shown below, where where S is substitution, P is permutation and K is key

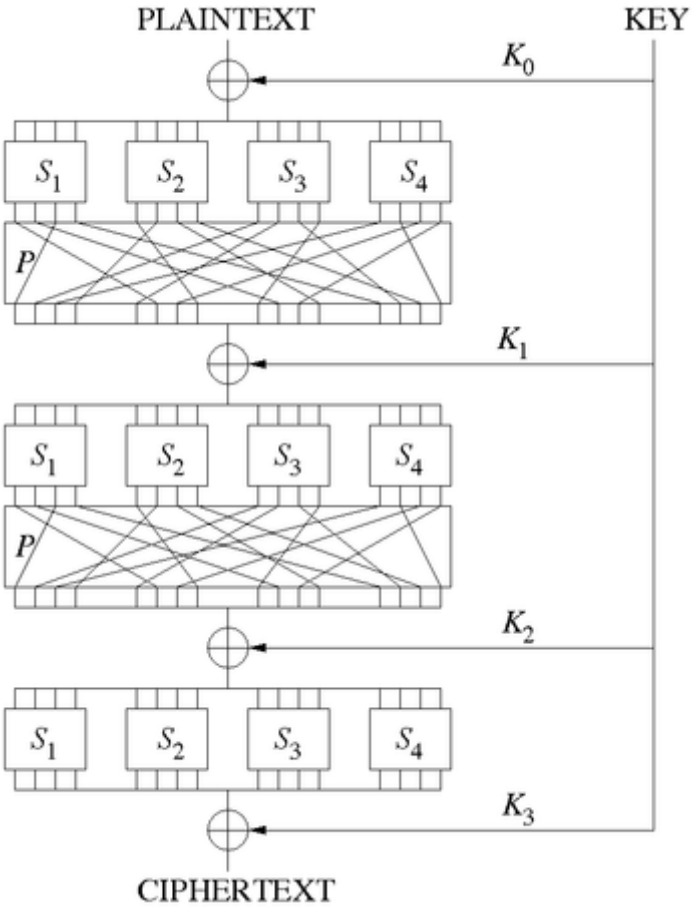


Figure 13. The structure of SPN

Feistel Network is one type of block cipher structure, which in the balanced Feistel network, the plaintext is divided into 2 equal parts which will be processed differently. The plaintext block is divided into two blocks, namely L and R. L and R for the i-th round are defined in the equations below, where F is Round Function for encryption or decryption and K is key

$$L_i + 1 = R_i \quad (3)$$

$$R_i + 1 = L_i \oplus F(R_i, K_i) \quad (4)$$

After that, Ciphertext is defined as $(R_n + 1, L_n + 1)$ where n is the number of rounds for the block cipher. The structure of Feistel Network is shown below.

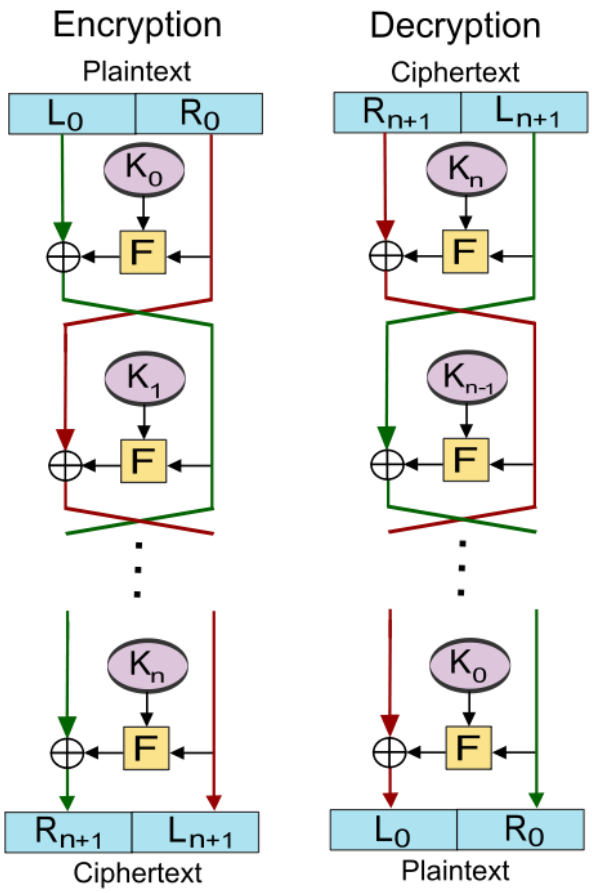


Figure 13. The structure of Feistel Network

2.4. Key Dependent Substitution

Key-Dependent Substitution is substitution which is affected by the key. Key-Dependent Substitution can be done using the Key-Dependent S-box, which is generated using an S-box generation algorithm based on the encryption key.

2.5. Key Dependent Permutation

Key-Dependent Permutation is permutation which is affected by the key. Key-Dependent Permutation can be done using the Key-Dependent P-box, which is generated using a P-box generation algorithm based on the encryption key.

2.6. Shannon's Diffusion and Confusion Principle

Shannon's Diffusion and Confusion Principle were terms identified by Claude Shannon in his 1945 classified report *A Mathematical Theory of Cryptography* [1]. They are two basic principles of block cipher design.

Diffusion means spreading the influence of a plaintext bit on the ciphertext to hide the statistical characteristics of a plaintext. In other words, a difference of one input bit can cause a significant change in ciphertext. A simple way that can be done to produce diffusion is the transpositions or permutation of plaintext at a certain level (byte / bit).

Confusion hides the relationship between plaintext and ciphertext. Confusion can be generated through substitution.

Simple operations such as substitution and permutation when performed multiple times on a plaintext block can produce good confusion and diffusion which implies a better level of security for the block cipher. This is what underlies the iterated cipher design for a block cipher.

3. The Proposed ITMFR Algorithm

ITFMR is designed to implement iterated cipher, transposition, substitution, and other algorithm by applying Shannon's Confusion and Diffusion principles. ITFMR uses 64-bit blocks, 64-bit keys, and 8 rounds. For each round, the algorithm will iterate transposition of block, substitute the block, and multiplying the block with the key and raising it to the power of the number of current ongoing rounds, each process will be done using the key obtained from that round.
The main components of ITMFR are:

1. Key-Dependent Iteration of Transposition
2. Key-Dependent Substitution

ITFMR also uses another operations, that is xor, multiplication, and exponentiation.

3.1. Specification

The specifications of ITFMR are:

1. Block size: 64-256 bits
2. Key size: 64-256 bits
3. Number of rounds: 8
4. Structure: Feistel Network

Figure 14 below describe how ITFMR works.

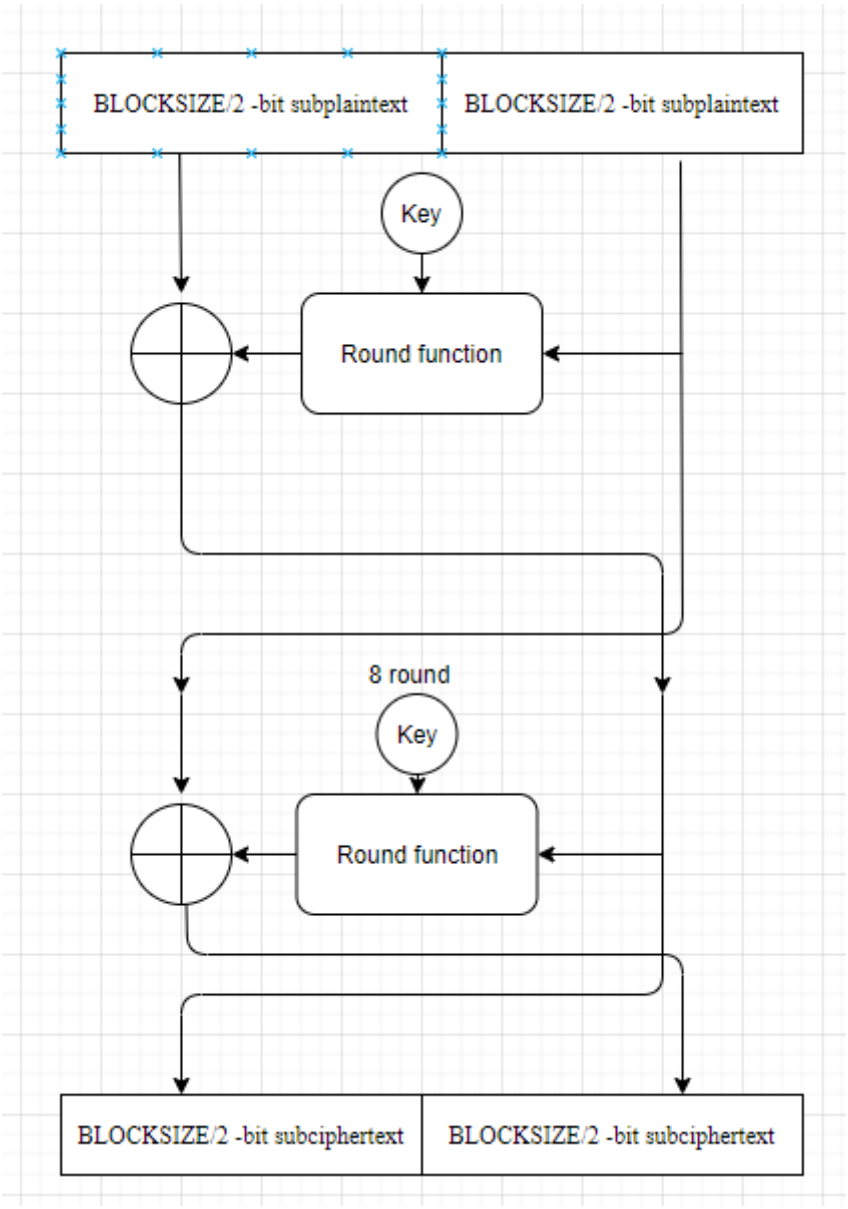


Figure 13. The Feistel Network structure of ITFMR

3.2. Key-Dependent Iteration of Transposition

ITFMR will iterate transposition of block n times, with n is the total of number 1 bit of the transposed block. Each transposition will have different keys starting from 0 to n. Let m be the key of the transposition process, we will make the bits of block into a matrix of m x (number of bits / m). The ciphered block will then be formed by combining the bit from each row 1 to m from each column 1 to (number of bits / m).

For example, let the block be 10101111 and the key be 2. The rows of the matrix form of the block will then be 10, 10, 11, 11. Taking each bit from the all the rows from row 1 to 4 and column 1 and combining it with each bit from column 2, we obtain 11110011 as the ciphered block.

In Python the implementation of this step is

key_iterate = 0

```
for i in range(len(key)):
    key_iterate = key_iterate + int(key[i])

for i in range(key_iterate):
    key_transpose = (key_iterate % len(block))
    while(len(block) % key_transpose != 0):
        key_transpose = key_transpose + 1
    new_block = ''
    matrix_block = [block[i:i+ key_transpose] for i in range(0, len(block), key_transpose)]
    for i in range(key_transpose):
        for j in range(len(matrix_block)):
            new_block = new_block + matrix_block[j][i]
    block = new_block
```

Note that the key of transposition processed will only be the factors of block length.

3.3. Key-Dependent Substitution

ITFMR will generate different S-Box table for each round based on the key from that round. ITFMR will use Rijndael S-Box as the base table as shown in Figure 6. ITFMR will generate new table based on the method described by Hosseinkhani and Javadi[9]. The generated table will be used in ITFMR to substitute every 8 bits from the block with the corresponding value on the table. The number of row will be taken from the first two and the last two bits from the 8-bit part of the block and the number of column will be taken from the third to sixth bits from the 8-bit part of the block. For example, 11001100 will be substituted with the value from row 1100, which is 12 in decimal, and column 0011, which is 3 in decimal. From the Rijndael table, 11001100 will be substituted as 37 in hex or 110111 in binary.

In Python the implementation of this step is

```
sbox = generateKeyDependentSBox(key)
eightbits_array = [block[i:i+8] for i in range(0, len(block), 8)]
for i in range(len(eightbits_array)):
    row = binaryToInteger(eightbits_array [i][:2] + eightbits_array [i][6:8])
    column = binaryToInteger(eightbits_array [i][3:6])
    eightbits_array [i] = integerToBinary(sbox[16*row+column]).split('b')[1]
block = ''.join(eightbits_array)
```

Note that the block that will be substituted in this step has already gone through the iteration of transposition step as explained before.

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

Figure 6. Rijndael S-Box[10]

3.4. Res function

ITFMR also add another step to make sure that the algorithm can be more secure. In this step, the algorithm will multiply the integer value of the block with the integer value of the key. The result will then be raised to the power of the number of current rounds conducted. The result will then be converted back to binary.

In Python the implementation of this method is

```
key = binaryToInteger(key)
block = binaryToInteger(block)

res = pow((block * key), NFeistelRound)
res = integerToBinary(res)
```

Note that the block used in this step has already gone through the substitution process as explained before.

3.5. Round Function

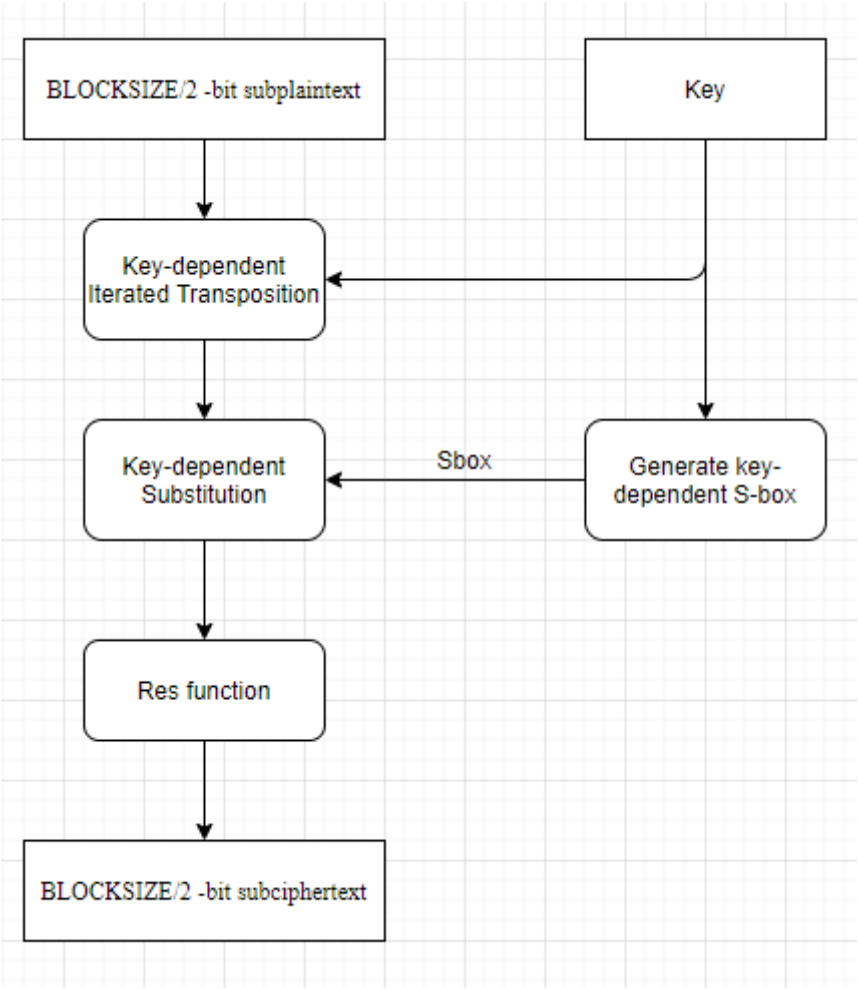


Figure 14. The Round Function structure of ITFMR

From the feistel network described above, the previous four steps in the algorithm play the role of transformation function (round function) *f* in the feistel network with the number of rounds is 8. Feistel network enables us to use the same function, that is *f*, for encryption and decryption.

4. Experiment & Result Analysis

4.1. Performance Analysis

ITFMR is implemented in Python 3.8.5 and tested on ACER Nitro AN515-52 with specifications:

1. Operating System: Windows 10 Home Single Language 64-bit
2. Processor: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
3. Memory: 8192 MB RAM

Experiment was done using three different plaintext, that is 2 block (32 bytes), 1000 blocks (16 KB), and 2000 blocks (32 KB). Each plaintext will be encrypted and decrypted using three different modes, that is ECB, CBC, and counter mode. The time taken to encrypt or decrypt will be measured in ms. The results of the experiment are shown in Table 1.

Table 1. Experiment Result

Mode	2 block (ms)	1000 blocks (ms)	2000 blocks (ms)
ECB (Encryption)	189	100549	158748
ECB (Decryption)	205	96500	160877

CBC (Encryption)	268	131738	230422
CBC (Decryption)	337	132301	231065
Counter (Encryption)	178	97193	171813
Counter (Decryption)	175	95976	193916

From the result above, we find that the ITFMR algorithm still run relatively slow since our algorithm use steps that yield large amount of possible combinations of ciphertext. But for a relatively small and important text, for example in messaging applications, ITFMR proved to be secure to use. The examples of plaintext, ciphertext, and key are shown in the table below.

Table 2. Example of Plaintext, Ciphertext, Master Key, and Expanded Key

Plaintext (in hex)	Ciphertext (in hex)	Master Key	Expanded Key
I, Giorno Giovanna, have a dream (49 2c 20 47 69 6f 72 6e 6f 20 47 69 6f 76 61 6e 6e 61 2c 20 68 61 76 65 20 61 20 64 72 65 61 6d)	q¼Pº²Dô5TÂÂÎQPç¨U jT;æ×{² (71 bc 90 51 8f 50 ba b2 44 f4 02 35 54 c2 c2 ce 51 50 e7 13 a8 55 6a 54 01 3b e6 d7 12 7b b2 15)	Bucciarati	33707e66483cafebb67 079430d0d10df20c299 501461130e3ccc36f14 765e3a1 df7e70e5021544f4834 bbee64a9e3789febc4b e81470df629cad6ddb0 3320a5c

4.2. Security Analysis

4.2.1. Brute-Force Attack.

For 1 block plaintext, since we use at least 64-bit block, there will be 64 possible keys for iteration of transposition. There will be 7 possible keys for transposition. For substtution, there will be 16*16 possible substitution for 8 bits of block, so there will be 256 raised to the power of 8 possible substitution for 1 block. Lastly, the last step of algorithm makes for 2 raised to the power of 128*8 possible results. The total amount of possible combinations makes it impossible for current technologies to analyze it within thousands of years.

4.2.2. Linear Cryptanalysis

The use of S-Box in ITFMR makes it harder for linear cyptanalysis since S-Box added non-linear characteristic to the algorithm. Linear cyptanalysis tries to find linear equation that has high bias that connects plaintext, ciphertext, and key. The high amount of possible combinations in S-Box makes harder to do that.

4.2.3. Differential Cryptanalysis

The use of S-Box in ITFMR makes it harder for linear cyptanalysis since S-Box added non-linear characteristic to the algorithm. Differential cyptanalysis tries to find certain differential output that often appears given certain differential input. The high amount of possible combinations in S-Box makes harder to do that.

5. Conclusion and Future Works

In this paper, we developed new block cipher algorithm that is impervious to any cryptanalysis attacks. ITFMR algorithm is secure since there are so many possible combinations for the ciphertext. Unfortunately, it still takes much time to encrypt the plaintext or decrypt the ciphertext. Future works can be conducted to make the algorithm more efficient and thus runs faster.

6. References

[1]"Information Theory and Entropy". *Model Based Inference in the Life Sciences: A Primer on Evidence*. Springer New York. 2008-01-01. pp. 51–82.

[2]Munir R 2020 *Review Beberapa Block Cipher dan Stream Cipher (Bagian 1: DES)* http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Review-beberapa-block-cipher-dan-stream-cipher-2020-bagian1.pdf (accessed on 23 October 2020)

[3]Munir R 2020 *Review Beberapa Block Cipher dan Stream Cipher (Bagian 4: Advanced Encryption Standard (AES))* http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Review-beberapa-block-cipher-dan-stream-cipher-2020-bagian4.pdf (accessed on 23 October 2020)

[4] R. C. Merkle, "Fast Software Encryption Functions", Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp 476-501, 1990.

[5] R. Zhang, L. Chen, "A Block Cipher using Key-Dependent S-Box and P-Boxes", Industrial Electronics, ISIE 2008. IEEE International Symposium, pp. 1463-1468, 2008.

[6] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall dan N. Ferguson, "Twofish : A 128-bit Block Cipher", AES Submission.

[7] B. Schneier (1993). "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)". Fast Software Encryption, Cambridge Security Workshop Proceedings. Springer-Verlag: 191–204.

[8] Munir R 2020 *Review Beberapa Block Cipher dan Stream Cipher (Bagian 3:* Block Cipher*)* http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Review-beberapa-block-cipher-dan-stream-cipher-2020-bagian3.pdf (accessed on 23 October 2020)

[9]Hosseinkhani R and Javadi H H S 2012 *Using Cipher Key to Generate Dynamic S-Box in AES Cipher System* (International Journal of Computer Science and Security vol 6)

[10]Wikipedia *Rijndael S-box* https://en.wikipedia.org/wiki/Rijndael_S-box (accessed on 23 October 2020)