Application of Boolean Algebra in AES-128 CBC Encryption Algorithm

Fityatul Haq Rosyidi - 13523116[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]FityatulHaqRosyidi25@gmail.com, 13523116@std.stei.itb.ac.id

***Abstract*—This paper examines the application of Boolean algebra in the AES encryption algorithm, focusing on the use of the XOR operation. The XOR operation is essential for securing the encryption process, particularly in CBC mode where previous ciphertext or the initialization vector (IV) is XORed with the current plaintext, enhancing the algorithm's resistance to cryptographic attacks. Additionally, the paper presents a Python implementation of the AES-ECB algorithm, showcasing basic encryption and decryption of text. The study highlights how Boolean operations, particularly XOR, contribute to the robustness of the AES algorithm.**

***Keywords*—AES, Boolean algebra, XOR , CBC mode**

I. Introduction

Data security is one of the most crucial aspects in the digital era, especially with the increasing threats to sensitive information stored and transmitted over networks. The Advanced Encryption Standard (AES) is a symmetric encryption algorithm that has become the international standard for protecting data with a high level of security. AES is designed to provide robust security by utilizing computational processes that involve manipulating bits and bytes of data. One approach that supports this process is the application of Boolean algebra, which allows data transformation to be more efficient and secure through basic logical operations such as AND, OR, and XOR.

At each stage of the algorithm, Boolean algebra concepts are used to manipulate data bits with high precision, creating data combinations that are difficult to predict by unauthorized parties. One approach to enhance security in AES is to use the Cipher Block Chaining (CBC) mode of operation. In this mode, each plaintext block is first operated with the result of the previous block's encryption using XOR before it is encrypted. This operation creates a chained encryption between blocks, ensuring that if there is any pattern in the plaintext, it cannot be detected.

II. Theoretical Framework

A. Boolean Algebra

Boolean algebra is a branch of mathematics that focuses on logical operations and the relationships between binary variables. These variables only have two values, namely 0 and 1, which generally represent the conditions of false and true in digital logic.

The basic concept of Boolean algebra was first proposed by English mathematician George Boole in 1854. However, the practical applications of Boolean algebra were not fully recognized for a long time after its introduction, both in mathematics and engineering. Later, in 1938, Claude Shannon, a communication expert, utilized and refined Boole's concept. [1]

The fundamental operations in Boolean algebra include NOT, AND, and OR. The NOT operation is a negation that inverts the logical value: if the input is 1, the output is 0, and vice versa. The AND operation (conjunction) results in True only if all inputs are True, whereas if any input is False, the result is False. Conversely, the OR operation (disjunction) produces True if one or more inputs are True, and only results in False if all inputs are False.

In addition to these basic operations, there are derived operations such as NAND, NOR, XOR, and XNOR. NAND (NOT AND) is an AND operation followed by a NOT operation, so the result is True except when all inputs are True. NOR (NOT OR) is an OR operation followed by a NOT operation, producing True only if all inputs are False. XOR (Exclusive OR) results in True if exactly one of the inputs is True, but not both. Conversely, XNOR (Exclusive NOR) is the negation of XOR, producing True if both inputs have the same value.

These operations are governed by various laws in Boolean algebra, such as the identity law, complement law, idempotent law, commutative law, associative law, and distributive law. Today, Boolean algebra plays a crucial role not only in logic but also in various other fields such as probability theory, information theory, set theory, cryptography, electronics, and more.

B. Cryptography

Cryptography refers to the process of encoding or disguising data to ensure that only the intended recipient, possessing the correct key, can access and understand the information. The term originates from the Greek words “kryptós,” meaning hidden, and “graphein,” meaning to write. Thus, cryptography literally means "hidden writing," but in practice, it is the secure transmission of information. [2]

The origins of cryptography date back to ancient Egyptian hieroglyphics, showcasing early methods of concealing information. Over time, the practice has evolved significantly, integrating advanced computing, mathematics, and engineering

to develop highly sophisticated algorithms and ciphers for safeguarding digital information in the modern world.

Today, cryptography underpins various encryption protocols that protect data, such as 128-bit and 256-bit encryption, Secure Sockets Layer (SSL), and Transport Layer Security (TLS). These technologies secure a wide range of digital activities, including protecting passwords, emails, online transactions, and sensitive financial operations.

Cryptographic methods come in different forms, tailored to specific needs. The simplest is symmetric key cryptography, where data is encrypted with a secret key, which is then sent to the recipient alongside the encoded message for decryption. However, this method is vulnerable because an intercepted key allows a third party to decrypt the message.

To address this limitation, asymmetric cryptography—or the "public key" system—was developed. In this approach, each user has a pair of keys: a public key and a private key. A sender encrypts a message using the recipient's public key, which can only be decrypted by the recipient's private key. This ensures that even if a message is intercepted, it cannot be decoded without access to the private key.

Cryptography plays a crucial role in cybersecurity, providing an extra layer of protection for data and users. It ensures privacy, maintains confidentiality, and helps safeguard information from cybercriminals. In practical terms, cryptography serves several purposes:

- **Confidentiality:** It guarantees that only the intended recipient can access and interpret the information, preserving the privacy of communications and data.
- **Data Integrity:** Cryptography ensures that data remains unaltered during transmission. Any attempt to modify the information would leave detectable evidence, as seen with digital signatures.
- **Authentication:** It verifies the identities of parties involved and the source or destination of the data.
- **Non-repudiation:** By using cryptographic methods like digital signatures or email tracking, senders are held accountable for their communications and cannot deny having sent a message.

Cryptography encompasses a wide range of processes, making its definitions understandably broad. This diversity arises from the various cryptographic algorithms available, each offering different levels of security depending on the nature of the information being transmitted. Below are the three primary types of cryptographic methods:

1. **Symmetric Key Cryptography:** This method is named for its use of a single shared key between the sender and receiver for both encryption and decryption. Examples include the Data Encryption Standard (DES) and Advanced Encryption Standard (AES). The main challenge with symmetric cryptography lies in securely sharing the key between the communicating parties.
2. **Asymmetric Key Cryptography:** This more secure approach involves two keys for each user: a public key and a private key. The sender uses the recipient's public key to encrypt the message, and the recipient uses their private key to decrypt it. Because the private key is kept secret, only the intended recipient can access the information. The RSA algorithm is the most widely used example of asymmetric cryptography.
3. **Hash Functions:** Unlike symmetric and asymmetric cryptography, hash functions do not use keys. Instead, they generate a unique hash value of fixed length based on the input data. This value acts as a unique identifier and is used for data encryption. Hash functions are commonly employed in operating systems for tasks like securing passwords.

The key distinction between symmetric and asymmetric encryption is that symmetric encryption uses a single shared key, while asymmetric encryption relies on a pair of keys—one public and one private—for added security.

C. Symmetric Encryption

Symmetric encryption is a method of encrypting data where the same key is used for both encoding and decoding the information. Until the introduction of the first asymmetric ciphers in the 1970s, symmetric encryption was the only available cryptographic technique.

In general, any cipher that relies on a single secret key for both encryption and decryption is classified as symmetric.

For instance, if the algorithm substitutes letters with numbers, both the sender and the recipient must share the same mapping table. The sender uses this table to encrypt the message, while the recipient uses it to decrypt it.

However, such basic ciphers are relatively easy to break. For example, if the frequency distribution of letters in a language is known, the most frequent letters can be matched with the most common numbers or symbols in the ciphertext, gradually revealing meaningful words. With the advent of computers, breaking such ciphers became trivial, rendering these methods obsolete.

Modern symmetric algorithms are considered secure only if they satisfy certain criteria:

- The encrypted data should not exhibit the same statistical patterns as the original data (for example, the most frequent symbols in the plaintext and the ciphertext should not be identical).
- The cipher must be nonlinear, meaning no detectable patterns or regularities exist in the encrypted data that would allow someone with access to both plaintext and ciphertext to trace the transformation.

To meet these requirements, most modern symmetric ciphers combine **substitution** (replacing portions of the plaintext, like letters, with other data, such as numbers, based on a specific rule or mapping table) and **permutation** (shuffling parts of the plaintext according to a rule). These operations are alternated and repeated in cycles, with each complete cycle referred to as a **round**.

Symmetric encryption algorithms can be classified into two categories based on their operational method:

- **Block ciphers**
- **Stream ciphers**

Block ciphers encrypt data in fixed-size blocks (e.g., 64, 128 bits, or other sizes depending on the algorithm). If the data or its final portion is smaller than the block size, the algorithm adds extra symbols, known as **padding**, to complete the block.

Examples of modern block ciphers include:

- AES
- GOST 28147-89
- RC5

- Blowfish
- Twofish

Stream ciphers use an **additive cipher**, where each bit of data is modified using the corresponding bit from a pseudorandom keystream (a sequence of numbers generated from a key) that matches the length of the message. Typically, the bits of the source data are compared with the keystream bits using the XOR logical operation (which returns 0 if the bits are the same and 1 if they differ).

Stream ciphers are used in algorithms like:

- RC4
- Salsa20
- HC-256
- WAKE

Symmetric algorithms are generally faster and require fewer resources than asymmetric algorithms. Most symmetric ciphers are believed to be resistant to attacks by **quantum computers**, which could theoretically threaten asymmetric encryption.

The primary weakness of symmetric encryption lies in the key exchange. Since both the sender and the recipient need the same key for encryption and decryption, this key must be transmitted securely. If sent over an unprotected channel, it could be intercepted. To mitigate this, many systems use asymmetric algorithms to encrypt and securely exchange the key.

Many modern services use symmetric encryption to protect data, often alongside asymmetric encryption. For example, instant messaging apps use these ciphers to secure messages, with the symmetric key typically being sent through asymmetric encryption. Similarly, video streaming services use symmetric encryption to secure audio and video streams. In the **Transport Layer Security (TLS)** protocol, symmetric encryption ensures the confidentiality of transmitted data.

However, symmetric encryption is not suitable for creating digital signatures and certificates because the secret key must be shared, which undermines the purpose of an electronic signature — the ability to verify authenticity without needing access to the owner's key.

III. Advanced Encryption Standard (AES)

A. Definition

Encryption plays a crucial role in today's digital landscape, promoting security and privacy. When the AES (Advanced Encryption Standard) algorithm replaced the Data Encryption Standard (DES) in 2001 as the global encryption standard, it addressed many of the limitations of its predecessor. AES was seen as the future of encryption for everyday applications, and it has successfully met the expectations set for it. Additionally, AES continues to evolve and improve.

When the DES algorithm was developed and standardized, it was suitable for the technology of that era. However, as computational power increased, breaking the DES encryption became faster and easier over time. To meet the growing need for stronger security, a more robust encryption algorithm was required, one with longer key sizes and stronger ciphers. While triple DES was introduced to address these issues, it did not become widely adopted due to its slower performance. This led to the creation of AES to overcome these challenges.

AES (also known as the Rijndael algorithm) is a symmetric block cipher with a 128-bit block size. It encrypts these blocks using keys of 128, 192, or 256 bits. After encrypting the individual blocks, AES combines them to form the final ciphertext.

AES uses a substitution-permutation network (SP network) model, which involves a series of operations like substitutions (replacing inputs with specific outputs) and permutations (rearranging bits).

Some key features of AES include:

1. **SP Network:** AES operates on an SP network structure, unlike the DES algorithm, which uses a Feistel cipher structure.
2. **Key Expansion:** AES begins with a single key and then expands it into multiple keys used in each encryption round.
3. **Byte Data:** AES processes data in byte-sized chunks rather than bit-sized chunks, treating the 128-bit block as 16 bytes during encryption.
4. **Key Length:** The number of rounds in AES encryption depends on the key size. A 128-bit key undergoes 10 rounds, a 192-bit key undergoes 12 rounds, and a 256-bit key undergoes 14 rounds.

B. Algorithm

To understand how AES operates, it is important to grasp how information is transferred through its various stages. Each block of data, comprising 16 bytes, is represented as a 4x4 matrix, where every cell contains a single byte of data. [3]

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figure 1. State Array
Source : Writer's Archive

This matrix is called the *state array*. Similarly, the initial key is expanded into (n+1) keys, where *n* represents the number of encryption rounds. For example, with a 128-bit key, there are 10 rounds, resulting in the generation of 11 keys in total.

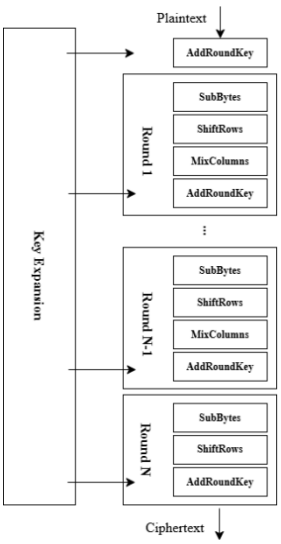


Figure 2. AES Algorithm Steps
Source : Writer's Archive

The outlined steps are performed sequentially on each block. Once all the individual blocks are successfully encrypted, they are combined to create the final ciphertext. The process includes the following steps:

- **Add Round Key**: The data stored in the *state array* is XORed with the initial key (K0). The resulting state array serves as input for the next step.

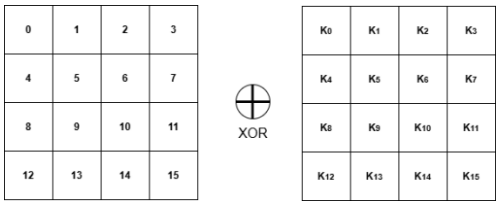


Figure 3. Add Round Key
Source : Writer's Archive

- **Sub-Bytes**: Each byte in the *state array* is converted to hexadecimal and split into two equal parts. These parts represent rows and columns, which are then used to look up corresponding values in the substitution box (S-Box), producing a new *state array*.

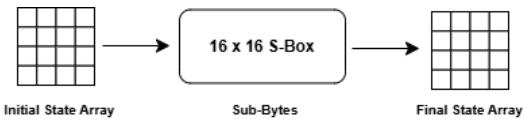


Figure 4. Sub-Bytes
Source : Writer's Archive

- **Shift Rows**: Row elements in the matrix are rearranged. The first row remains unchanged, the second row is shifted one position to the left, the third row is shifted two positions to the left, and the fourth row is shifted three positions to the left.

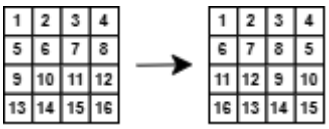


Figure 5. Shift Rows
Source : Writer's Archive

- **Mix Columns**: Each column of the *state array* is multiplied by a constant matrix to produce a new column. After all columns are transformed, the updated *state array* proceeds to the next step. However, this step is omitted during the final encryption round.

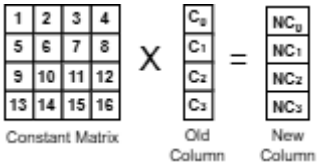


Figure 6. Mix Columns
Source : Writer's Archive

C. AES Modes

Encryption algorithms can be categorized based on the input type into two main types: block ciphers and stream ciphers. A block cipher is an encryption algorithm that processes a fixed-size input (e.g., *b* bits) and produces an output ciphertext of the same size (*b* bits). If the input exceeds *b* bits, it is divided into smaller blocks for processing.

Block Cipher Modes of Operation define the methods used to securely encrypt and decrypt large data sets with a block cipher. Block ciphers operate on fixed-size data blocks (e.g., 128 bits) rather than processing data bit by bit. To handle data larger than a single block while maintaining security and efficiency, various modes of operation are applied. Below are some common modes of operation:

Electronic Code Book (ECB)

The Electronic Code Book is the simplest mode of operation for a block cipher. It works by directly encrypting each block of plaintext, producing corresponding blocks of encrypted ciphertext. When a message is larger than *b* bits, it is divided into smaller blocks, and the encryption process is repeated for each block.

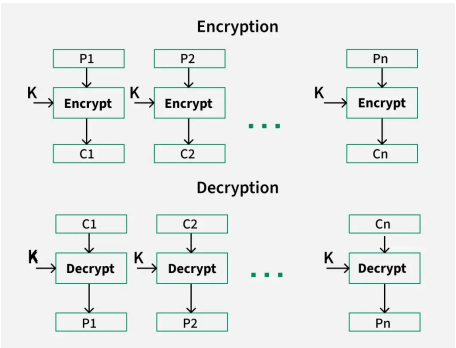


Figure 7. ECB
Source : Block Cipher modes of Operation - GeeksforGeeks

The Electronic Code Book (ECB) mode offers advantages such as enabling parallel encryption of data blocks, making it a faster encryption method, and its simplicity in implementation. However, it also has significant drawbacks, including vulnerability to cryptanalysis due to the direct correlation between plaintext and ciphertext. Additionally, identical plaintext blocks result in identical ciphertext blocks, potentially exposing patterns and compromising data security.

Cipher Block Chaining (CBC)

The Cipher Block Chaining (CBC) improves upon the security limitations of the Electronic Code Book (ECB) mode. In CBC, each plaintext block is XORed with the ciphertext of the previous block before being encrypted, creating a dependency between consecutive blocks. In essence, the cipher block is generated by encrypting the result of the XOR operation between the current plaintext block and the previous ciphertext block.

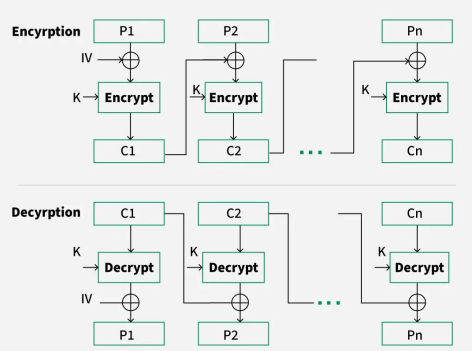


Figure 8. CBC
Source : Block Cipher modes of Operation - GeeksforGeeks

The Cipher Block Chaining (CBC) mode offers several advantages, including effective handling of input data larger than *b* bits, serving as a reliable authentication mechanism, and providing stronger resistance to cryptanalysis compared to ECB by obscuring patterns in the data. However, CBC has a drawback: it relies on the previous ciphertext block for both encryption and decryption, which complicates parallel processing.

Cipher Feedback Mode (CFB)

In this mode, the cipher output is fed back into the encryption process for the subsequent block, incorporating specific features. Initially, an initialization vector (IV) is used for the first encryption. The output bits are then divided into two parts: *s* bits on the left and *b-s* bits on the right. The *s* bits are combined with the plaintext through an XOR operation, and the result is input into a shift register. This register shifts *b-s* bits to the left-hand side (*lhs*) and *s* bits to the right-hand side (*rhs*), continuing the process. The encryption and decryption procedures follow a similar approach, utilizing encryption algorithms in both cases.

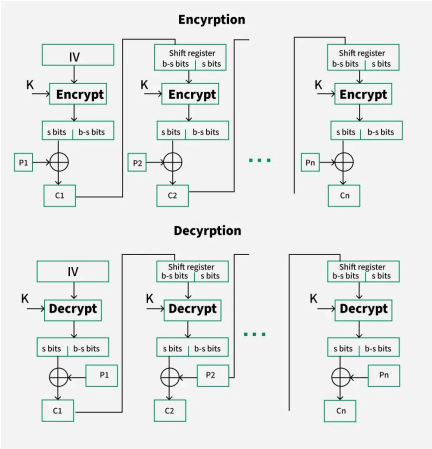


Figure 9. CFB
Source : Block Cipher modes of Operation - GeeksforGeeks

The Cipher Feedback (CFB) mode offers advantages such as making cryptanalysis challenging due to data loss from the use of shift registers and its ability to process data streams of any size. However, it shares similar limitations with CBC mode, including the inability to support block loss or simultaneous encryption of multiple blocks. While decryption in CFB is parallelizable and resilient to loss, the mode is slightly more complex and prone to error propagation.

Output Feedback Mode (OFB)

The Output Feedback (OFB) mode operates similarly to the Cipher Feedback mode, with the key difference being that it uses the encrypted output as feedback rather than the XOR output of the cipher. In OFB, the entire block of bits is sent, rather than just selecting *s* bits. This mode provides strong resistance to bit transmission errors and reduces the dependency or relationship between the ciphertext and the plaintext.

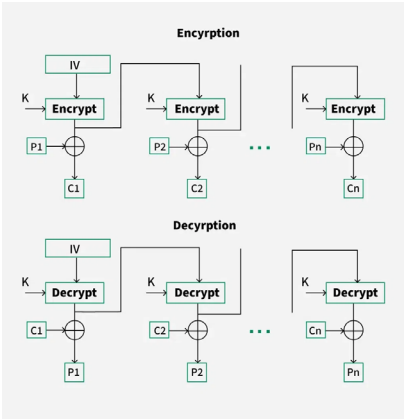


Figure 10. OFB
Source : Block Cipher modes of Operation - GeeksforGeeks

The Output Feedback (OFB) mode has the advantage of preventing bit errors from propagating, unlike Cipher Feedback (CFB), where a single bit error in a block can affect all subsequent blocks. This makes OFB more resilient to transmission errors. However, OFB has its drawbacks, including being more vulnerable to message stream modification attacks compared to CFB due to its operational method. Additionally, if

the keystream is reused, it compromises the security of the encryption.

Counter Mode (CTR)

The Counter Mode (CTR) is a straightforward block cipher implementation that relies on a counter. In this mode, a value generated by the counter is encrypted and then XORed with the plaintext to produce the ciphertext block. Since the CTR mode does not require feedback, it can be implemented in parallel, offering increased efficiency.

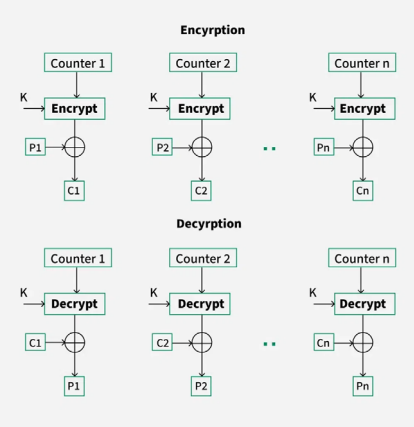


Figure 11. CTR
Source : Block Cipher modes of Operation - GeeksforGeeks

The Counter (CTR) mode offers the advantage of eliminating the direct relationship between plaintext and ciphertext, as each block uses a unique counter value, allowing the same plaintext to map to different ciphertext. It also enables parallel encryption, as outputs from previous stages are not chained like in CBC. However, a significant disadvantage of CTR is that it requires synchronization of the counter between both the transmitter and receiver. If synchronization is lost, the recovery of plaintext becomes inaccurate.

IV. IMPLEMENTATION

Python Languange was chosen for the implementation of this algorithm due to its effectiveness and ease of program design. The AES CBC algorithm is implemented from scratch, without using pre-existing AES classes such as those in the pyCryptodome library. The purpose of building this program from scratch is to deepen the understanding of the sequence of processes in the AES algorithm, including both encryption and decryption processes. The program repository can be accessed via the link provided in the appendix.

A. Key Expanding

```python
def expand_key(master_key):
    r_con = (
        0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
        0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
        0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
        0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
    )
    key_columns = bytes2matrix(master_key)
    iteration_size = len(master_key) // 4
    i = 1
    while len(key_columns) < (N_ROUNDS + 1) * 4:
        word = list(key_columns[-1])
        if len(key_columns) % iteration_size == 0:
            word.append(word.pop(0))
            word = [s_box[b] for b in word]
            word[0] ^= r_con[i]
            i += 1
        elif len(master_key) == 32 and len(key_columns) % iteration_size == 4:
            word = [s_box[b] for b in word]
        word = bytes(i^j for i, j in zip(word, key_columns[-iteration_size]))
        key_columns.append(word)
    return [key_columns[4*i : 4*(i+1)] for i in range(len(key_columns) // 4)]
```

Figure 12. Key Expanding
Source : Writer's Archive

The expand_key function is used to perform key expansion, generating 11 round keys that are each the result of XOR operations. Each round key is used alternately in each round of the encryption process.

B. Encryption

```python
def add_round_key(s, k):
    return [[sss^kkk for sss,kkk in zip(ss,kk)] for ss,kk in zip(s,k)]

def shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]

def sub_bytes(s, sbox=s_box):
    state = list(map(lambda x: sbox[x], sum(s, [])))
    return [list(state[i:i+4]) for i in range(0, len(state), 4)]

def mix_single_column(a):
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)

def mix_columns(s):
    for i in range(4):
        mix_single_column(s[i])
```

Figure 13. Encrypt Stages Functions
Source : Writer's Archive

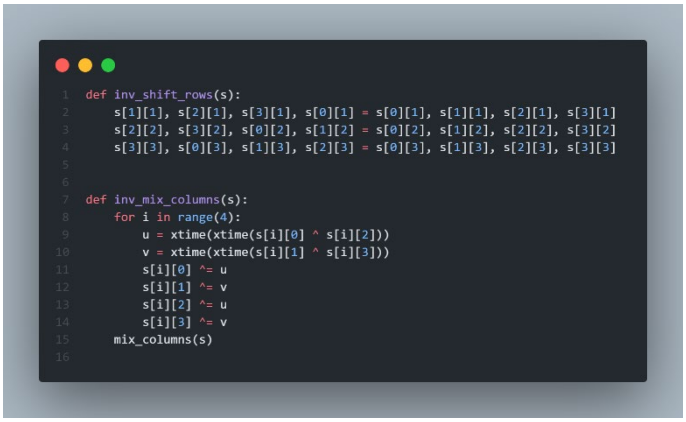
There are four main stages in the encryption process: add_round_key, shift_rows, sub_bytes, and mix_columns. Each of these stages is used of the encryption process.

```python
def single_block_encrypt(key, plaintext) :
    round_keys = expand_key(key)
    state = bytes2matrix(plaintext)
    state = add_round_key(state, round_keys[0])
    for i in range(1, N_ROUNDS) :
        state = sub_bytes(state, s_box)
        shift_rows(state)
        mix_columns(state)
        state = add_round_key(state, round_keys[i])
    state = sub_bytes(state, s_box)
    shift_rows(state)
    state = add_round_key(state, round_keys[10])
    ciphertext = bytes(sum(state, []))
    return ciphertext

def cbc_encrypt(key, iv, plaintext):
    plaintext = pad(plaintext, 16)
    pt_blocks = [plaintext[i:i+16] for i in range(0, len(plaintext), 16)]
    ct_blocks = ["" for i in range(len(pt_blocks))]
    ct_blocks[0] = single_block_encrypt(key, bytes([x ^ y for x,y in zip(iv, pt_blocks[0])]))
    for i in range(1, len(pt_blocks)):
        ct_blocks[i] = single_block_encrypt(key, bytes([x ^ y for x,y in zip(ct_blocks[i-1], pt_blocks[i])]))
    ciphertext = b''.join(ct_blocks)

    return ciphertext
```

Figure 14. Encrypt Main Functions
Source : Writer's Archive

The single_block_encrypt function is used for encrypting a single block that contains 16 bytes of data. Each individual block will be linked with others if the plaintext exceeds 16 bytes, using the Cipher Block Chaining (CBC) mode, which is implemented in the cbc_encrypt function.

C. Decryption



```python
def inv_shift_rows(s):
    s[1][1], s[2][1], s[3][1], s[0][1] = s[0][1], s[1][1], s[2][1], s[3][1]
    s[2][2], s[3][2], s[0][2], s[1][2] = s[0][2], s[1][2], s[2][2], s[3][2]
    s[3][3], s[0][3], s[1][3], s[2][3] = s[0][3], s[1][3], s[2][3], s[3][3]


def inv_mix_columns(s):
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v
    mix_columns(s)
```

Figure 15. Decrypt Stages Function
Source : Writer's Archive

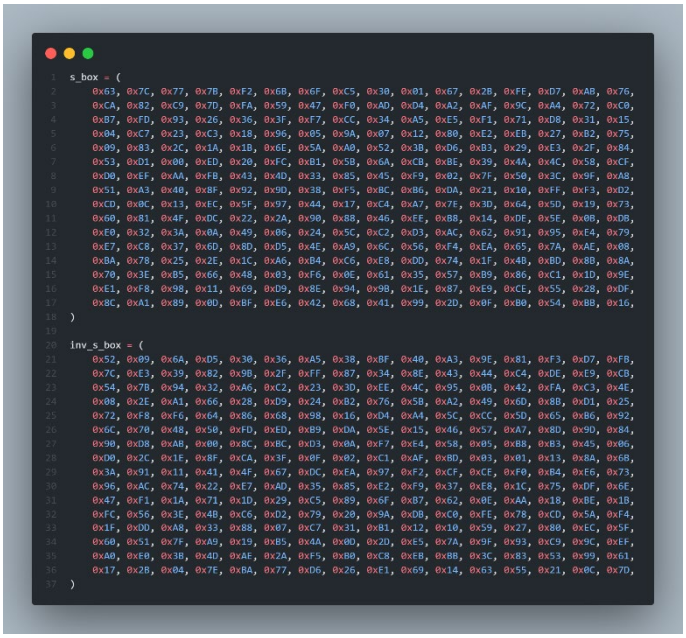
In the decryption process, only two new functions are required: inv_shift_rows and inv_mix_columns. As their names suggest, these functions are used to reverse the shift_rows and mix_columns processes. Additionally, the sub_bytes and add_round_key processes do not need to be reversed, as they are symmetric operations.

```python
def single_block_decrypt(key, ciphertext):
    round_keys = expand_key(key)
    state = bytes2matrix(ciphertext)
    state = add_round_key(state, round_keys[10])
    for i in range(N_ROUNDS - 1, 0, -1) :
        inv_shift_rows(state)
        state = sub_bytes(state, inv_s_box)
        state = add_round_key(state, round_keys[i])
        inv_mix_columns(state)
    inv_shift_rows(state)
    state = sub_bytes(state, inv_s_box)
    state = add_round_key(state, round_keys[0])
    plaintext = bytes(sum(state, []))
    return plaintext

def cbc_decrypt(key, iv, ciphertext):
    ct_blocks = [ciphertext[i:i+16] for i in range(0, len(ciphertext), 16)]
    pt_blocks = ["" for i in range(len(ct_blocks))]
    pt_blocks[0] = bytes([x ^ y for x,y in zip(iv, single_block_decrypt(key, ct_blocks[0]))])
    for i in range(1, len(ct_blocks)):
        pt_blocks[i] = bytes([x ^ y for x,y in zip(ct_blocks[i-1], single_block_decrypt(key, ct_blocks[i]))])
    plaintext = b''.join(pt_blocks)
    plaintext = unpad(plaintext, 16)

    return plaintext
```

Figure 16. Decrypt Main Function
Source : Writer's Archive

Similar to the encryption process, the single_block_decrypt function is used for decrypting a single block that contains 16 bytes of data. Likewise, the cbc_decrypt function is used to decrypt ciphertext longer than 16 bytes using the CBC mode.

D. Constants



```python
s_box = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
)

inv_s_box = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
)
```

Figure 17. Constants
Source : Writer's Archive

There are two constant matrices in this program: s_box and inv_s_box. The s_box is a constant used for byte substitution during the sub_bytes step. The values in the state array are replaced with the corresponding values from the s_box. On the other hand, the inv_s_box is a constant used to reverse the byte substitution during decryption.

V. CONCLUSION

This paper explores the use of Boolean algebra in the AES encryption algorithm, particularly within the CBC (Cipher Block Chaining) mode. The key Boolean operation employed in AES is XOR, which plays a crucial role in the AddRoundKey

step, where the key undergoes repeated XOR operations. Additionally, in the CBC mode, the previous plaintext is XORed with either the initialization vector (IV) or the prior ciphertext. This technique strengthens data security by increasing encryption complexity and mitigating the effectiveness of cyberattacks.

This paper also includes an implementation of the AES-ECB algorithm in Python, demonstrating a basic encryption and decryption program for text.

VI. Appendix

Github Repository : FityatulhaqRosyidi/Simple-AES-128bit-CBC-Implementation: Advanced Encryption Standard (AES) Implementation, focused on Cipher Block Chaining (CBC) Mode

VII. Acknowledgment

I would like to express my deepest gratitude to God Almighty, for His grace and blessings, allowing me to complete this paper. I also wish to sincerely thank all the individuals who have provided support and assistance in the preparation of this paper. Special thanks go to Dr. Ir. Rinaldi Munir, M.T. and Dr. Ir. Rila Mandala, M.Eng., Ph.D., the lecturer for the IF1220 Discrete Mathematics course. I also extend my gratitude and encouragement to my fellow students in the Department of Informatics Engineering, whose support has kept me motivated and made me feel supported throughout this process.

Lastly, I hope this paper can provide valuable insights to its readers and contribute to the advancement of knowledge, particularly in the fields of Cryptography and Discrete Mathematics.

References

[1] G. Boole, *Sejarah Aljabar Boolean*, Academia.edu. [Online]. Available: https://www.academia.edu/25398298/Sejarah_aljabar_boolean. [Accessed: Jan. 8, 2025].

[2] "Apa itu Kriptografi," Dewaweb. [Online]. Available: https://www.dewaweb.com/blog/apa-itu-kriptografi/#Pengertian_Kriptografi. [Accessed: Jan. 8, 2025].

[3] "AES Challenge," CryptoHack. [Online]. Available: https://cryptohack.org/challenges/aes/. [Accessed: Jan. 8, 2025].

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Januari 2025

Fityatul Haq Rosyidi
13523116