BAB III

METODOLOGI

3.1. Metodologi Penelitian

Penelitian ini merupakan penelitian eksperimen kuantitatif, yaitu pendekatan penelitian yang melibatkan manipulasi terhadap satu atau lebih variabel bebas untuk mengamati pengaruhnya terhadap variabel terikat, di mana hasilnya diukur dan dianalisis secara numerik menggunakan metrik yang terstandarisasi (Takona, 2024). Pendekatan ini dipilih karena penelitian ini memenuhi tiga karakteristik utama penelitian eksperimen, yaitu pertama adanya treatment berupa modifikasi algoritma AES-128 melalui penggantian fungsi *MixColumns* dengan mekanisme *dynamic key-dependent bit permutation* berbasis *logistic chaotic map*, kedua, adanya baseline pembanding yang digunakan sebagai acuan perbandingan hasil, dan terakhir, adanya pengukuran dampak dari treatment tersebut melalui metrik kuantitatif yang terukur, yaitu Kecepatan Enkripsi, Kecepatan Dekripsi, *Avalanche effect*, *Throughput* , Entropi Shannon, *Key sensitivity*, *Frequency test*, *Runs test*, dan *Chi-Square test*. Data yang dihasilkan seluruhnya berupa data numerik yang dianalisis secara kuantitatif.

3.2. Lingkungan Pengujian

Pada penelitian ini, seluruh pengujian pada algoritma AES-128 standar maupun yang telah dimodifikasi yang dijadikan pembanding dilakukan menggunakan perangkat keras dan perangkat lunak dengan spesifikasi yang sama sebagai berikut: laptop yang digunakan dilengkapi dengan prosesor AMD Ryzen 5 5500U dengan

Radeon Graphics yang memiliki kecepatan dasar 2.10 GHz, disertai dengan 16 GB RAM (15.3 GB usable) yang mendukung kelancaran pemrosesan data dalam jumlah besar. Penyimpanan yang digunakan adalah 512 GB SSD, yang memungkinkan akses data yang lebih cepat selama pengujian. Sistem operasi yang digunakan adalah Windows 11, yang memastikan kompatibilitas dengan berbagai alat pengembangan dan perangkat lunak modern. Penelitian ini juga menggunakan Python 3.13 sebagai bahasa pemrograman utama untuk mengimplementasikan algoritma dan mengukur performa enkripsi dan dekripsi. Dengan spesifikasi tersebut, diharapkan dapat diperoleh hasil yang optimal dalam pengujian performa algoritma yang dimodifikasi.

Tabel 3.1. Spesifikasi Perangkat Keras dan Perangkat Lunak

Prosessor	AMD Ryzen 5 5500U with Radeon Graphics (2.10 GHz)
Ram	16.0 GB (15.3 GB usable)
Penyimpanan	512 GB
Sistem Operasi	Windows 11
Bahasa Pemrograman	Python 3.13

3.3.Pembangkitan Tabel Permutasi Dinamis

Tahap pembangkitan tabel permutasi dinamis dilakukan pada bagian pra-proses, dengan menggunakan SHA-256 kunci enkripsi yang kemudian diproses melalui *logistic chaotic map* dan *Fisher-Yates shuffle* untuk menghasilkan tabel permutasi yang bergantung pada kunci. Gambar 3.1 menunjukkan proses pembangkitan tabel permutasi dinamis yang digunakan sebagai pengganti fungsi *MixColumns* pada

algoritma AES-128. Proses ini diawali dengan kunci enkripsi 128-bit yang diproses menggunakan fungsi hash SHA-256 untuk menghasilkan digest sepanjang 256-bit. Tahap ini bertujuan untuk meningkatkan keacakan dan menyebarkan distribusi bit kunci. Selanjutnya, 64-bit pertama dari hasil hash diekstraksi dan dinormalisasi ke dalam rentang $(0, 1)$untuk membentuk nilai awal x_0, yang akan digunakan sebagai seed dalam sistem *Chaos*. Nilai awal tersebut kemudian diproses menggunakan *logistic chaotic map* dengan parameter r = 3,99 untuk menghasilkan deret bilangan chaotic. Iterasi dilakukan sebanyak 50 kali untuk menghilangkan efek transient dan memastikan sistem berada dalam kondisi *Chaos* yang stabil. Deret bilangan yang dihasilkan memiliki sifat sensitif terhadap kondisi awal dan sulit diprediksi, sehingga cocok digunakan sebagai sumber pengacakan dalam proses pembangkitan permutasi.

Selanjutnya, deret chaotic tersebut digunakan sebagai input pada algoritma *Fisher-Yates shuffle* untuk mengacak array indeks dan membentuk tabel permutasi bit. Proses ini diulang hingga dihasilkan empat tabel permutasi utama, yaitu BP1, BP2, BP3, dan BP4, beserta tabel inversnya untuk proses dekripsi. Dengan pendekatan ini, tabel permutasi yang dihasilkan bersifat dinamis dan bergantung pada kunci enkripsi, Sehingga diharapkan dapat membantu mempertahankan kualitas difusi *ciphertext* pada algoritma dengan menghilangkan kelemahan pada tabel permutasi yang bersifat statis.

3.3.1. Pseudocode Pembentukan BP1–BP4 dan Tabel Invers

Untuk memperjelas rancangan implementasi algoritma yang diusulkan, tahapan pembangkitan tabel permutasi dinamis dijelaskan dalam bentuk

pseudocode. Pseudocode digunakan agar alur komputasi dapat dipahami secara sistematis tanpa bergantung pada sintaks bahasa pemrograman tertentu.

generate_bp_dan_tabel_invers

```
Input : Kunci enkripsi K berukuran 128-bit
Output : BP1, BP2, BP3, BP4
         InvBP1, InvBP2, InvBP3, InvBP4

K ← Key Input
H ← SHA256(K)
S64 ← ambil 64-bit pertama dari H
S ← konversi S64 menjadi bilangan bulat positif
x0 ← (S + 1) / (2^64 + 1)
r ← 3.99
x ← x0

for i ← 1 to 50 do
    x ← r × x × (1 - x)
end for

BPCollection ← array kosong
InvBPCollection ← array kosong
for t ← 1 to 4 do
    BP ← [0, 1, 2, 3, 4, 5, 6, 7]
    for i ← 7 downto 1 do
        x ← r × x × (1 - x)
        j ← floor(x × (i + 1))
        if j > i then
            j ← i
        end if
        tukar BP[i] dengan BP[j]
    end for
    InvBP ← array kosong berukuran 8
    for i ← 0 to 7 do
        InvBP[BP[i]] ← i
    end for
    simpan BP ke BPCollection[t]
    simpan InvBP ke InvBPCollection[t]
end for

BP1 ← BPCollection[1]
BP2 ← BPCollection[2]
BP3 ← BPCollection[3]
BP4 ← BPCollection[4]
InvBP1 ← InvBPCollection[1]
InvBP2 ← InvBPCollection[2]
InvBP3 ← InvBPCollection[3]
InvBP4 ← InvBPCollection[4]

return BP1, BP2, BP3, BP4, InvBP1, InvBP2, InvBP3, InvBP4
```

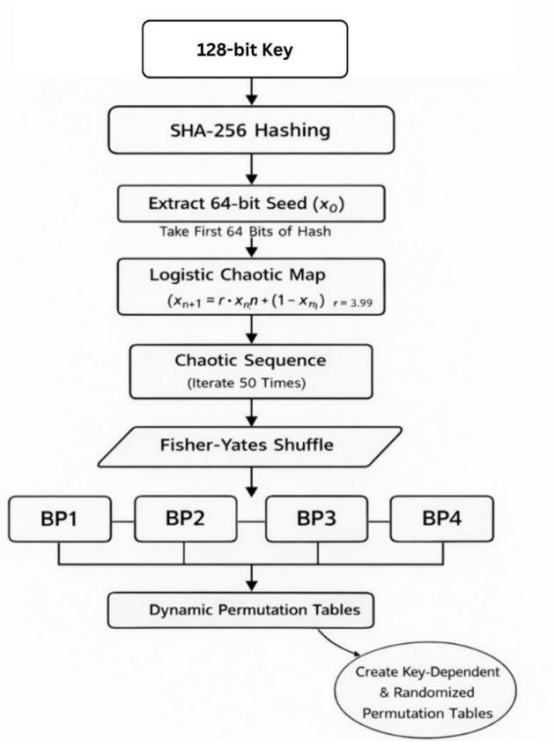
3.3.2. Mekanisme Penerapan BP1–BP4 pada State AES

BP1, BP2, BP3, dan BP4 adalah empat tabel permutasi bit yang digunakan untuk menggantikan fungsi *MixColumns* dalam AES termodifikasi. Setiap tabel diterapkan pada byte yang berbeda dalam satu kolom state AES. BP1 digunakan untuk byte baris pertama, BP2 untuk byte baris kedua, BP3 untuk byte baris ketiga, dan BP4 untuk byte baris keempat. Dengan demikian, setiap kolom state yang terdiri dari empat byte diproses menggunakan empat pola permutasi bit yang berbeda.

Pada blok 128-bit, state AES berbentuk matriks 4×4 byte. Oleh karena itu, byte ke-1 sampai byte ke-4 diproses sebagai kolom pertama, byte ke-5 sampai byte ke-8 sebagai kolom kedua, byte ke-9 sampai byte ke-12 sebagai kolom ketiga, dan byte ke-13 sampai byte ke-16 sebagai kolom keempat. Pada setiap kolom, pola penerapan BP tetap sama, yaitu BP1 untuk byte baris pertama, BP2 untuk byte baris kedua, BP3 untuk byte baris ketiga, dan BP4 untuk byte baris keempat. Mekanisme ini membuat setiap posisi byte dalam kolom memperoleh pola permutasi yang berbeda, sehingga membantu mempertahankan kualitas difusi *ciphertext*.

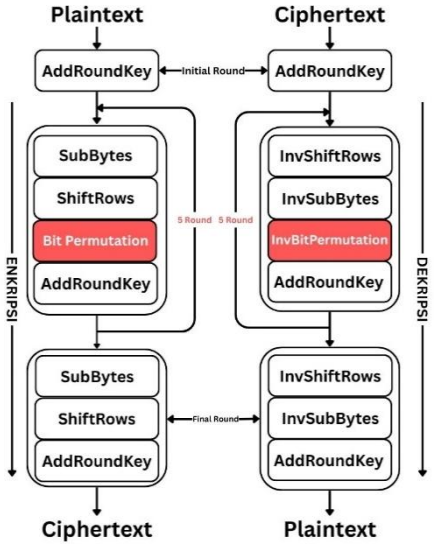
BP1–BP4 dibangkitkan dari kombinasi SHA-256, *logistic chaotic map*, dan *Fisher-Yates shuffle*, sehingga tabel permutasi bersifat bergantung pada kunci enkripsi. Untuk proses dekripsi, digunakan tabel invers InvBP1–InvBP4 untuk membalikkan permutasi bit dan mengembalikan *ciphertext* menjadi plaintext. Pengurangan jumlah *round* dari 10 *round* menjadi 6 *round* tidak hanya bertujuan untuk meningkatkan kecepatan enkripsi dan dekripsi, tetapi juga berfungsi sebagai strategi untuk mengurangi beban komputasi total akibat adanya mekanisme

tambahan pada tahap pra-proses. Pada algoritma yang diusulkan, pembangkitan tabel permutasi dinamis berbasis *logistic chaotic map* memang menimbulkan *overhead* awal karena melibatkan proses hashing kunci, pembentukan deret *Chaos*, dan *Fisher-Yates shuffle*. Namun, *overhead* tersebut dikendalikan dengan dua cara, yaitu tabel permutasi hanya dibangkitkan satu kali untuk setiap kunci enkripsi dan jumlah *round* AES dikurangi sehingga operasi transformasi yang dilakukan berulang pada setiap blok menjadi lebih sedikit. Dengan demikian, *reduced round* berperan sebagai mekanisme kompensasi komputasi agar penambahan proses dinamis tidak menyebabkan algoritma menjadi lebih lambat secara keseluruhan. Kemudian, *Overhead* pada algoritma yang diusulkan bersifat terbatas pada tahap pra-proses, bukan terjadi berulang pada setiap blok atau setiap *round*, sehingga pengaruhnya terhadap total waktu enkripsi diharapkan tetap terkendali.



Gambar 3.1. Pembangkitan tabel permutasi dinamis

3.4. Proses Enkripsi-Dekripsi



Gambar 3.2. Usulan Proses Enkripsi-Dekripsi

Pada algoritma AES-128 termodifikasi yang diusulkan, jumlah *round* dikurangi dari 10 *round* menjadi 6 *round*. Oleh karena itu, proses key expansion menghasilkan tujuh *round* key, yaitu RK_0 sampai RK_6. RK_0 digunakan pada tahap initial Add*Round*Key sebelum proses *round* dimulai, sedangkan RK_1 sampai RK_6 digunakan pada proses *round* enkripsi. Skema enkripsi terdiri atas initial *round*, lima main *round*, dan satu final *round*. Initial *round* dilakukan dengan menerapkan Add*Round*Key menggunakan RK_0. Selanjutnya, *round* ke-1 sampai *round* ke-5 berperan sebagai main *round* yang masing-masing terdiri atas SubBytes, ShiftRows, *Dynamic bit permutation* sebagai pengganti *MixColumns*, dan Add*Round*Key menggunakan RK_1 sampai RK_5. *Round* ke-6 berperan sebagai final *round* yang hanya terdiri atas SubBytes, ShiftRows, dan Add*Round*Key menggunakan RK_6 tanpa *Dynamic bit permutation*. Dengan demikian, *Dynamic bit permutation* hanya diterapkan pada main *round*, sedangkan final *round* mengikuti

prinsip AES yang tidak menerapkan transformasi pengganti *MixColumns* pada *round* terakhir.

Pemilihan 6 *round* pada algoritma modifikasi ini tidak dilakukan secara arbitrer, melainkan didasarkan pada studi ablasi dan analisis kriptanalisis fundamental terhadap arsitektur AES. Berdasarkan dokumentasi resmi desain AES, Salah satu penelitian mengonfirmasi bahwa difusi penuh (*full diffusion*) telah tercapai hanya dalam dua putaran, dan 6 *round* merupakan ambang batas (*threshold*) struktural di mana algoritma mencapai kompleksitas maksimal untuk menahan serangan analitis utama, seperti *Square Attack* (Daemen & Rijmen, 2002). Hal ini sejalan dengan studi kriptanalisis ekstensif yang membuktikan secara eksplisit bahwa eksploitasi matematis paling efektif terhadap AES standar hanya mampu menembus hingga batas 6 hingga 7 putaran (Ferguson dkk., 2001). Oleh karena itu, sisa putaran hingga genap 10 *round* pada AES-128 sejatinya diimplementasikan semata-mata sebagai security margin (margin keamanan cadangan) untuk mengantisipasi kelemahan dari fungsi *MixColumns* yang bersifat statis dan linier (Biryukov dkk., 2010).

Mengingat penelitian ini telah mereduksi kerenxtanan linieritas tersebut dengan menginjeksi dynamic key-dependent bit permutation berbasis logistic chaotic map yang memiliki kompleksitas pengacakan jauh lebih tinggi per putarannya, maka pemangkasan security margin ke batas teoretis 6 *round* merupakan justifikasi matematis yang optimal; komputasi menjadi sangat efisien tanpa mendegradasi Avalanche Effect di bawah standar steady-state yang telah matang pada putaran ke-6.

3.4.1. Representasi Data, Segmentasi Blok, dan Padding

AES-128 bekerja pada blok data 128-bit, yang berarti setiap blok yang diolah oleh algoritma adalah berukuran 16 byte (128 bit). Oleh karena itu, untuk mengadaptasi data yang lebih besar, file teks harus dibagi menjadi blok-blok berukuran 128-bit.

Setiap file teks yang akan dienkripsi terlebih dahulu dibagi menjadi beberapa blok 128-bit. Jika panjang file tidak merupakan kelipatan dari 128-bit, maka blok terakhir akan diisi dengan padding. Proses segmentasi ini penting untuk memastikan bahwa seluruh data dapat diproses oleh algoritma AES-128. Untuk memastikan bahwa data dapat dibagi menjadi blok 128-bit yang tepat, padding digunakan untuk menambahkan byte tambahan pada data yang tidak memenuhi panjang blok. Padding yang digunakan dalam penelitian ini adalah PKCS#7 padding. Jika panjang data yang dienkripsi tidak kelipatan 16 byte (128-bit), maka padding akan ditambahkan untuk memenuhi ukuran blok 128-bit. Padding yang ditambahkan terdiri dari byte dengan nilai yang sama dengan jumlah byte yang dibutuhkan. Misalnya, jika blok terakhir memiliki panjang 14 byte, maka akan ditambahkan 2 byte padding dengan nilai 0x02 (hexadecimal) untuk mencapai panjang 16 byte. Jika panjang data sudah kelipatan 16 byte, maka satu blok penuh (16 byte) dengan nilai 0x10 akan ditambahkan sebagai padding.

3.4.1.1. Pseudocode Padding PKCS#7

Padding_pkcs#7

```
Input  : DataBytes
         BlockSize = 16
Output : ResultBytes

PanjangData ← panjang DataBytes
```

```
if PanjangData mod BlockSize ≠ 0 then
    SisaByte ← PanjangData mod BlockSize
    JumlahPadding ← BlockSize - SisaByte
    PaddingBytes ← byte dengan nilai JumlahPadding sebanyak
JumlahPadding kali
    ResultBytes ← gabungkan DataBytes dan PaddingBytes
else
    JumlahPadding ← nilai byte terakhir dari DataBytes
    if JumlahPadding ≥ 1 and JumlahPadding ≤ BlockSize then
        PaddingBytes ← ambil JumlahPadding byte terakhir
dari DataBytes
        if seluruh byte pada PaddingBytes bernilai
JumlahPadding then
            ResultBytes ← hapus JumlahPadding byte terakhir
dari DataBytes
        else
            JumlahPadding ← BlockSize
            PaddingBytes ← byte dengan nilai JumlahPadding
sebanyak JumlahPadding kali
            ResultBytes ← gabungkan DataBytes dan
PaddingBytes
        end if
    else
        JumlahPadding ← BlockSize
        PaddingBytes ← byte dengan nilai JumlahPadding
sebanyak JumlahPadding kali
        ResultBytes ← gabungkan DataBytes dan PaddingBytes
    end if
end if
return ResultBytes
```

3.4.2. Mode Operasi Cipher Block Chaining (CBC)

Mode operasi yang digunakan dalam penelitian ini adalah Cipher Block Chaining (CBC). Mode ini dipilih karena file yang diuji lebih besar dari satu blok 128-bit, sehingga setiap blok plaintext di-XOR dengan *ciphertext* dari blok sebelumnya sebelum dienkripsi. Pada blok pertama, digunakan Initialization Vector (IV) sepanjang 128-bit agar *ciphertext* tetap berbeda meskipun plaintext dan kunci yang digunakan sama. Dengan CBC, enkripsi setiap blok bergantung pada blok

sebelumnya, sehingga mengurangi pola data yang dapat dimanfaatkan dalam kriptanalisis.

3.4.2.1. Pseudocode XOR Dua Blok

XOR_dua_blok

```
Input  : BlockA berukuran 16 byte
         BlockB berukuran 16 byte
Output : ResultBlock berukuran 16 byte

ResultBlock ← array kosong berukuran 16 byte

for i ← 0 to 15 do
    ResultBlock[i] ← BlockA[i] XOR BlockB[i]
end for

return ResultBlock
```

3.4.3. Proses Enkripsi File Multi-Blok

Proses enkripsi file multi-blok dimulai dengan membaca file teks sebagai data byte, kemudian membaginya menjadi blok-blok 128-bit. Setelah proses segmentasi selesai, blok terakhir diberi padding menggunakan PKCS#7 apabila ukurannya belum mencapai 16 byte. Selanjutnya, IV digunakan pada blok pertama dengan melakukan operasi XOR antara IV dan blok plaintext pertama. Hasil XOR tersebut kemudian dienkripsi menggunakan algoritma AES-128 termodifikasi yang telah dirancang dalam penelitian ini. Untuk blok kedua dan seterusnya, setiap blok plaintext di-XOR terlebih dahulu dengan *ciphertext* dari blok sebelumnya, kemudian hasilnya dienkripsi menggunakan algoritma yang sama. Proses ini dilakukan secara berurutan hingga seluruh blok plaintext selesai dienkripsi. Seluruh *ciphertext* dari setiap blok kemudian digabungkan menjadi satu keluaran *ciphertext* file.

proses_enkripsi_file

```
Input  : PlaintextFile
         Kunci K berukuran 128-bit
Output : CiphertextFile

PlainBytes ← baca seluruh isi PlaintextFile sebagai byte
if panjang K ≠ 16 byte then
    tampilkan pesan "Kunci tidak valid"
    hentikan proses
end if

PanjangData ← panjang PlainBytes
SisaByte ← PanjangData mod 16
if SisaByte = 0 then
    JumlahPadding ← 16
else
    JumlahPadding ← 16 - SisaByte
end if

PaddingBytes ← byte dengan nilai JumlahPadding sebanyak
JumlahPadding kali
PaddedPlainBytes ← gabungkan PlainBytes dan PaddingBytes
BlokPlaintext ← bagi PaddedPlainBytes menjadi blok-blok
berukuran 16 byte
BP1, BP2, BP3, BP4, InvBP1, InvBP2, InvBP3, InvBP4 ←
GenerateDynamicPermutationTables(K)
RoundKey ← KeyExpansionAES128(K)
RK0 ← RoundKey[0]
RK1 ← RoundKey[1]
RK2 ← RoundKey[2]
RK3 ← RoundKey[3]
RK4 ← RoundKey[4]
RK5 ← RoundKey[5]
RK6 ← RoundKey[6]
IV ← bangkitkan nilai acak berukuran 16 byte
PreviousBlock ← IV
CiphertextCollection ← array kosong

for setiap PlainBlock dalam BlokPlaintext do
    InputBlock ← XOR(PlainBlock, PreviousBlock)
    State ← ubah InputBlock menjadi matriks 4×4 byte
    State ← AddRoundKey(State, RK0)
    for round ← 1 to 5 do
        State ← SubBytes(State)
```

```
            State ← ShiftRows(State)
            for kolom ← 0 to 3 do
                ByteInput ← State[0][kolom]
                BitInput ← ubah ByteInput menjadi array 8 bit
                BitOutput ← array kosong berukuran 8 bit
                for i ← 0 to 7 do
                    BitOutput[i] ← BitInput[BP1[i]]
                end for
                State[0][kolom] ← ubah BitOutput menjadi byte

                ByteInput ← State[1][kolom]
                BitInput ← ubah ByteInput menjadi array 8 bit
                BitOutput ← array kosong berukuran 8 bit
                for i ← 0 to 7 do
                    BitOutput[i] ← BitInput[BP2[i]]
                end for
                State[1][kolom] ← ubah BitOutput menjadi byte

                ByteInput ← State[2][kolom]
                BitInput ← ubah ByteInput menjadi array 8 bit
                BitOutput ← array kosong berukuran 8 bit
                for i ← 0 to 7 do
                    BitOutput[i] ← BitInput[BP3[i]]
                end for
                State[2][kolom] ← ubah BitOutput menjadi byte

                ByteInput ← State[3][kolom]
                BitInput ← ubah ByteInput menjadi array 8 bit
                BitOutput ← array kosong berukuran 8 bit
                for i ← 0 to 7 do
                    BitOutput[i] ← BitInput[BP4[i]]
                end for
                State[3][kolom] ← ubah BitOutput menjadi byte
            end for
            State ← AddRoundKey(State, RoundKey[round])
        end for

        State ← SubBytes(State)
        State ← ShiftRows(State)
        State ← AddRoundKey(State, RK6)
        CipherBlock ← ubah State menjadi array 16 byte
        simpan CipherBlock ke CiphertextCollection
        PreviousBlock ← CipherBlock
    end for
```

```
CiphertextBytes ← gabungkan IV dan seluruh CipherBlock dalam
CiphertextCollection
tulis CiphertextBytes ke CiphertextFile
return CiphertextFile
```

Pseudocode enkripsi file AES-128 termodifikasi menjelaskan tahapan pengamanan data mulai dari pembacaan plaintext hingga menghasilkan *ciphertext* file. Proses diawali dengan membaca isi file sebagai byte, kemudian dilakukan validasi panjang kunci agar sesuai dengan AES-128, yaitu 16 byte atau 128 bit. Setelah itu, data diberi PKCS#7 padding agar panjangnya menjadi kelipatan 16 byte, lalu dibagi menjadi blok-blok berukuran 128 bit. Sebelum proses enkripsi blok dilakukan, algoritma membangkitkan tabel permutasi dinamis BP1, BP2, BP3, dan BP4 berdasarkan kunci, serta melakukan key expansion untuk menghasilkan RK0 sampai RK6. Setiap blok plaintext diproses menggunakan mode CBC, yaitu di-XOR terlebih dahulu dengan IV pada blok pertama atau *ciphertext* blok sebelumnya pada blok berikutnya. Hasil XOR kemudian diproses melalui AES-128 termodifikasi, dimulai dari initial Add*Round*Key menggunakan RK0, dilanjutkan lima main *round* yang terdiri atas SubBytes, ShiftRows, *Dynamic bit permutation*, dan Add*Round*Key, kemudian diakhiri dengan final *round* yang hanya terdiri atas SubBytes, ShiftRows, dan Add*Round*Key menggunakan RK6. Setiap cipher block yang dihasilkan disimpan, kemudian seluruhnya digabungkan bersama IV untuk membentuk *ciphertext* file akhir.

3.4.4. Proses Dekripsi

Proses dekripsi dilakukan dengan urutan kebalikan dari proses enkripsi. *Ciphertext* dibagi kembali menjadi blok-blok 128-bit, kemudian setiap blok

didekripsi menggunakan kunci yang sama dan tabel invers dari *dynamic bit permutation*. Hasil dekripsi blok pertama di-XOR dengan IV untuk memperoleh plaintext blok pertama, sedangkan hasil dekripsi blok berikutnya di-XOR dengan *ciphertext* blok sebelumnya. Setelah seluruh blok berhasil didekripsi, padding pada blok terakhir dihapus sesuai aturan PKCS#7 sehingga plaintext asli dapat diperoleh kembali secara utuh. Susunan ini memastikan bahwa skema yang diusulkan bersifat invertible dan dapat digunakan untuk proses enkripsi maupun dekripsi file multi-blok.

3.4.4.1. Pseudocode Dekripsi

Proses_dekripsi_file

```
Input  : CiphertextFile
         Kunci K berukuran 128-bit
Output : PlaintextFile

CiphertextBytes ← baca seluruh isi CiphertextFile sebagai
byte
if panjang K ≠ 16 byte then
    tampilkan pesan "Kunci tidak valid"
    hentikan proses
end if

if panjang CiphertextBytes < 32 byte then
    tampilkan pesan "Ciphertext tidak valid"
    hentikan proses
end if

IV ← ambil 16 byte pertama dari CiphertextBytes
CipherBytes ← ambil sisa byte setelah IV

if panjang CipherBytes mod 16 ≠ 0 then
    tampilkan pesan "Ukuran ciphertext tidak valid"
    hentikan proses
end if

BlokCiphertext ← bagi CipherBytes menjadi blok-blok
berukuran 16 byte
```

```
BP1, BP2, BP3, BP4, InvBP1, InvBP2, InvBP3, InvBP4 ←
Generate Dynamic PermutationTables(K)
Round Key ← KeyExpansionAES128(K)
RK0 ← Round Key[0]
RK1 ← Round Key[1]
RK2 ← Round Key[2]
RK3 ← Round Key[3]
RK4 ← Round Key[4]
RK5 ← Round Key[5]
RK6 ← Round Key[6]
PreviousBlock ← IV
PlaintextCollection ← array kosong

for setiap CipherBlock dalam Blok Ciphertext do
    State ← ubah CipherBlock menjadi matriks 4×4 byte
    State ← Add RoundKey(State, RK6)
    State ← InvShiftRows(State)
    State ← InvSubBytes(State)

    for round ← 5 downto 1 do
        State ← Add RoundKey(State, RoundKey[round])

        for kolom ← 0 to 3 do
            ByteInput ← State[0][kolom]
            BitInput ← ubah ByteInput menjadi array 8 bit
            BitOutput ← array kosong berukuran 8 bit
            for i ← 0 to 7 do
                BitOutput[i] ← BitInput[InvBP1[i]]
            end for
            State[0][kolom] ← ubah BitOutput menjadi byte

            ByteInput ← State[1][kolom]
            BitInput ← ubah ByteInput menjadi array 8 bit
            BitOutput ← array kosong berukuran 8 bit
            for i ← 0 to 7 do
                BitOutput[i] ← BitInput[InvBP2[i]]
            end for
            State[1][kolom] ← ubah BitOutput menjadi byte

            ByteInput ← State[2][kolom]
            BitInput ← ubah ByteInput menjadi array 8 bit
            BitOutput ← array kosong berukuran 8 bit
            for i ← 0 to 7 do
                BitOutput[i] ← BitInput[InvBP3[i]]
```

```
            end for
            State[2][kolom] ← ubah BitOutput menjadi byte

            ByteInput ← State[3][kolom]
            BitInput ← ubah ByteInput menjadi array 8 bit
            BitOutput ← array kosong berukuran 8 bit
            for i ← 0 to 7 do
                BitOutput[i] ← BitInput[InvBP4[i]]
            end for
            State[3][kolom] ← ubah BitOutput menjadi byte
        end for

        State ← InvShiftRows(State)
        State ← InvSubBytes(State)
    end for

    State ← AddRoundKey(State, RK0)
    DecryptedBlock ← ubah State menjadi array 16 byte
    PlainBlock ← XOR(DecryptedBlock, PreviousBlock)
    simpan PlainBlock ke PlaintextCollection
    PreviousBlock ← CipherBlock
end for

PaddedPlainBytes ← gabungkan seluruh PlainBlock dalam
PlaintextCollection
JumlahPadding ← nilai byte terakhir dari PaddedPlainBytes

if JumlahPadding ≥ 1 and JumlahPadding ≤ 16 then
    PaddingBytes ← ambil JumlahPadding byte terakhir dari
PaddedPlainBytes
    if seluruh byte pada PaddingBytes bernilai
JumlahPadding then
        PlainBytes ← hapus JumlahPadding byte terakhir dari
PaddedPlainBytes
    else
        tampilkan pesan "Padding tidak valid"
        hentikan proses
    end if
else
    tampilkan pesan "Padding tidak valid"
    hentikan proses
end if

tulis PlainBytes ke PlaintextFile
```

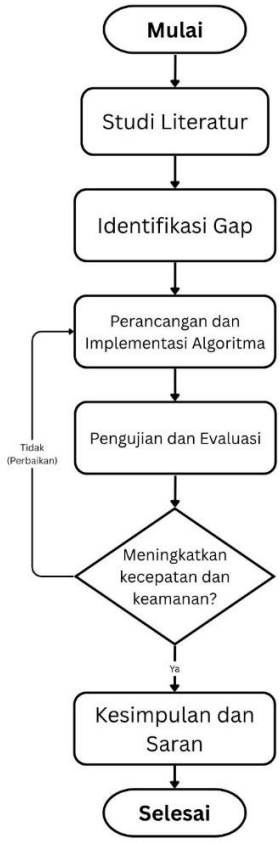
```
return PlaintextFile
```

Pseudocode dekripsi file AES-128 termodifikasi menjelaskan tahapan pengembalian *ciphertext* menjadi plaintext asli menggunakan kunci yang sama dengan proses enkripsi. Proses diawali dengan membaca *ciphertext* file sebagai byte, memvalidasi panjang kunci, memisahkan IV dari 16 byte pertama *ciphertext*, serta memastikan ukuran *ciphertext* valid dan dapat dibagi ke dalam blok-blok 16 byte. Selanjutnya, tabel permutasi dinamis dan tabel inversnya dibangkitkan ulang menggunakan kunci yang sama, sehingga InvBP1, InvBP2, InvBP3, dan InvBP4 dapat digunakan untuk membalikkan proses *Dynamic bit permutation*. Setiap blok *ciphertext* diproses melalui urutan transformasi kebalikan dari enkripsi, dimulai dengan Add*Round*Key menggunakan RK6, kemudian InvShiftRows dan InvSubBytes. Setelah itu, proses main *round* dilakukan secara mundur dari *round* ke-5 sampai *round* ke-1 dengan urutan Add*Round*Key, Inverse *Dynamic bit permutation*, InvShiftRows, dan InvSubBytes. Setelah seluruh transformasi blok selesai, Add*Round*Key menggunakan RK0 dilakukan untuk menghasilkan decrypted block, lalu hasilnya di-XOR dengan IV atau *ciphertext* blok sebelumnya sesuai aturan CBC. Semua plaintext block kemudian digabungkan, divalidasi padding-nya, dan PKCS#7 padding dihapus agar diperoleh plaintext file asli.

3.5.Alur Penelitian

Alur penelitian ini menjelaskan tahapan yang dilakukan secara sistematis dalam pelaksanaan penelitian, mulai dari studi literatur, identifikasi masalah, perancangan dan implementasi algoritma, hingga proses pengujian dan analisis hasil. Alur

penelitian disusun agar setiap tahap dapat dilaksanakan secara terstruktur, sehingga proses pengembangan algoritma dan evaluasi performanya dapat berjalan dengan jelas, terukur, dan dapat dipertanggungjawabkan. Dengan demikian, subbab ini menjadi acuan utama dalam menggambarkan urutan kerja penelitian dari awal hingga diperoleh hasil akhir yang sesuai dengan tujuan penelitian.



Gambar 3.3. Alur Penelitian

3.5.1. Studi Literatur

Studi literatur merupakan tahapan eksplorasi dan analisis area penelitian untuk memperoleh pemahaman serta dasar teori dalam mendukung penelitian. Studi literatur dilakukan dengan mencari informasi dari buku, hasil penelitian, dan

artikel ilmiah yang berkaitan dengan topik utama, mencakup modifikasi algoritma AES-128, *bit permutation*, *logistic chaotic map*, *Avalanche effect*, dan entropi Shannon. Sumber referensi yang digunakan bersumber dari berbagai jurnal bereputasi nasional maupun internasional terindeks. Indikator tahapan ini adalah diperolehnya pemahaman mendalam terhadap area penelitian sehingga dapat menghasilkan Bab I Pendahuluan dan Bab II Kajian Pustaka.

3.5.2. Identifikasi Masalah

Tahap identifikasi gap penelitian dilakukan berdasarkan hasil studi literatur yang telah dilakukan sebelumnya. Pemetaan dilakukan dengan membandingkan pendekatan, metode, hasil, dan keterbatasan dari penelitian terdahulu secara sistematis menggunakan tabel perbandingan. Dari hasil pemetaan ditemukan bahwa Baladhay dan De Los Reyes (2024) menggunakan tabel permutasi yang bersifat *fixed* dan sama untuk seluruh kunci enkripsi, sementara Salih dkk. (2020) membuktikan bahwa komponen yang bersifat statis dan publik dalam AES dapat dieksploitasi melalui kriptanalisis diferensial dan linear. Berdasarkan hasil kajian literatur, penelitian-penelitian sebelumnya umumnya hanya berfokus pada penggantian *MixColumns* atau penerapan *chaotic map* secara terpisah. Oleh karena itu, belum ditemukan pendekatan yang secara bersamaan mengintegrasikan penggantian *MixColumns* menggunakan *dynamic bit permutation* berbasis *chaotic map* pada AES. Celah penelitian tersebut menjadi landasan dan justifikasi penelitian ini. Indikator tahapan ini adalah teridentifikasinya research gap yang valid berdasarkan bukti ilmiah dari literatur yang telah dikaji.

3.5.3. Perancangan dan Implementasi Algoritma

Perancangan algoritma dilakukan berdasarkan gap penelitian yang telah diidentifikasi pada tahap sebelumnya. Rancangan algoritma mencakup dua komponen utama. Pertama, mekanisme pembangkitan tabel permutasi dinamis menggunakan *logistic chaotic map* dengan parameter r = 3,99 yang di-*seed* dari SHA-256 kunci enkripsi melalui algoritma *Fisher-Yates shuffle*, sehingga setiap kunci enkripsi yang berbeda menghasilkan tabel permutasi yang unik dan tidak dapat diprediksi. Kedua, prosedur enkripsi 6 *round* yang menggantikan *MixColumns* dengan *Dynamic bit permutation* menggunakan tabel yang dibangkitkan secara dinamis. Pada rancangan ini, RK_0 digunakan untuk initial Add*Round*Key, RK_1 sampai RK_5 digunakan pada lima main *round*, dan RK_6 digunakan pada final *round*. Setiap main *round* menerapkan SubBytes, ShiftRows, *Dynamic bit permutation*, dan Add*Round*Key secara berurutan, sedangkan final *round* hanya menerapkan SubBytes, ShiftRows, dan Add*Round*Key tanpa *Dynamic bit permutation*. Seluruh rancangan didokumentasikan dalam bentuk pseudocode dan flowchart yang mencakup alur enkripsi, alur dekripsi, dan alur pembangkitan tabel dinamis. Indikator tahapan ini adalah dihasilkannya rancangan algoritma yang lengkap dan siap untuk diimplementasikan.

Implementasi dilakukan terhadap algoritma yang telah dirancang menggunakan bahasa pemrograman Python. Pemilihan Python didasarkan pada kemudahan prototyping, ketersediaan library hashlib untuk fungsi SHA-256 yang digunakan dalam pembangkitan seed *logistic map*, serta kemudahan verifikasi operasi bitwise. Implementasi dilakukan secara from scratch tanpa menggunakan

library AES eksternal agar setiap komponen algoritma dapat dikontrol dan diverifikasi secara langsung. Komponen yang diimplementasikan meliputi key expansion standar AES, SubBytes dan inversnya, ShiftRows dan inversnya, Add*Round*Key, pembangkitan tabel permutasi dinamis melalui *logistic map* dan *Fisher-Yates shuffle*, serta *dynamic bit permutation* beserta prosedur inversnya. Keberhasilan implementasi diverifikasi dengan memastikan bahwa proses dekripsi dapat memulihkan *plaintext* asli secara sempurna dari *ciphertext* yang dihasilkan, serta memastikan bahwa kunci yang berbeda menghasilkan tabel permutasi yang berbeda. Indikator tahapan ini adalah dihasilkannya implementasi perangkat lunak yang berfungsi dengan benar dan terverifikasi.

3.5.4. Pengujian dan Evaluasi

Pengujian dan evaluasi dilakukan untuk mengetahui pengaruh penerapan *reduced round* dan *dynamic key-dependent bit permutation* berbasis *logistic chaotic map* terhadap performa dan kualitas hasil enkripsi pada algoritma AES-128 termodifikasi. Pengujian ini dilakukan dengan membandingkan algoritma yang diusulkan terhadap AES-128 standar dan algoritma penelitian terdahulu yang dijadikan sebagai *baseline*. Perbandingan tersebut bertujuan untuk menilai apakah modifikasi yang dilakukan mampu meningkatkan efisiensi proses enkripsi dan dekripsi tanpa menurunkan kualitas difusi pada *ciphertext*.

3.5.4.1.Data Uji

Data uji yang digunakan dalam penelitian ini berupa file teks berformat .txt. Penggunaan file teks dipilih karena AES memproses data sebagai rangkaian byte, sehingga isi dan struktur internal file tidak memengaruhi mekanisme dasar enkripsi.

Ukuran file yang digunakan terdiri atas tujuh variasi, yaitu: 10 KB, 20 KB, 30 KB, 40 KB, 50 KB, 100 KB, 1 MB. Pemilihan ukuran tersebut dilakukan untuk melihat performa algoritma pada beberapa tingkat ukuran data. Ukuran 10 KB sampai 50 KB digunakan untuk merepresentasikan file kecil, ukuran 100 KB digunakan sebagai transisi menuju file menengah, sedangkan ukuran 1 MB digunakan untuk menguji skalabilitas algoritma ketika jumlah blok data yang diproses meningkat. Dengan variasi tersebut, pengaruh ukuran file terhadap waktu enkripsi, waktu dekripsi, dan *Throughput* dapat diamati secara lebih jelas. Pemilihan ukuran hingga 1 MB didasarkan pada karakteristik AES sebagai block cipher yang memproses data secara berulang pada setiap blok 128 bit. Karena setiap blok diproses menggunakan rangkaian transformasi yang sama, pertambahan ukuran file pada dasarnya hanya menambah jumlah blok yang diproses tanpa mengubah mekanisme internal algoritma. Dengan demikian, pengujian hingga 1 MB dianggap memadai untuk merepresentasikan perilaku algoritma ketika jumlah blok meningkat dan untuk mengevaluasi skalabilitas kinerjanya.

3.5.4.2. Skenario Pengujian

Pengujian dilakukan terhadap lima jenis algoritma, yaitu AES-128 standar, AES-128 dengan *bit permutation* statis, AES-128 dengan *bit permutation* statis dan *reduced round*, AES dengan *Dynamic MixColumns* berbasis *chaotic map*, serta algoritma AES-128 termodifikasi yang diusulkan dalam penelitian ini. Seluruh algoritma pembanding diimplementasikan dan diuji pada lingkungan perangkat keras, perangkat lunak, bahasa pemrograman, ukuran data, serta jumlah pengulangan yang sama. Hal ini dilakukan agar hasil perbandingan waktu enkripsi,

waktu dekripsi, dan *Throughput* tidak dipengaruhi oleh perbedaan platform pengujian. Setiap ukuran file diuji sebanyak 20 kali untuk memperoleh nilai rata-rata yang lebih representatif dan meminimalkan pengaruh fluktuasi waktu eksekusi. Seluruh hasil pengujian dicatat dalam spreadsheet, kemudian dihitung nilai rata-ratanya berdasarkan setiap parameter pengujian. Data hasil pengujian selanjutnya divisualisasikan dalam bentuk grafik untuk memudahkan analisis perbandingan performa dan kualitas hasil enkripsi antar algoritma.

3.5.4.3. Parameter Pengujian

Parameter pengujian dalam penelitian ini terdiri atas enam metrik kuantitatif, yaitu:

1) Kecepatan Enkripsi

   Kecepatan enkripsi digunakan untuk mengukur waktu yang dibutuhkan algoritma dalam mengubah plaintext menjadi *ciphertext*. Parameter ini penting karena penelitian ini berfokus pada peningkatan efisiensi AES-128 melalui pengurangan jumlah *round* dan penggantian fungsi *MixColumns*. Formula pengukuran kecepatan enkripsi dapat dituliskan pada persamaan (3.1).

$$T_{enc} = t_{end} - t_{start} \tag{3.1}$$

2) Kecepatan Dekripsi

   Kecepatan dekripsi digunakan untuk mengukur waktu yang dibutuhkan algoritma dalam mengembalikan *ciphertext* menjadi plaintext. Parameter ini diperlukan karena algoritma kriptografi simetris tidak hanya harus efisien pada proses enkripsi, tetapi juga pada proses dekripsi. Formula pengukuran kecepatan dekripsi dapat dituliskan pada persamaan (3.2).

$$T_{dec} = t_{end} - t_{start} \quad (3.2)$$

3) *Throughput*

Throughput digunakan untuk mengukur jumlah data yang dapat diproses oleh algoritma dalam satuan waktu tertentu. Parameter ini digunakan agar evaluasi performa tidak hanya dilihat dari waktu proses, tetapi juga dari kemampuan algoritma dalam memproses ukuran data yang berbeda. Formula *Throughput* enkripsi dapat dituliskan pada persamaan (3.3.1), dan formula *Throughput* dekripsi dituliskan pada persamaan (3.3.2).

$$Throughput_{enc} = \frac{\text{Ukuran data}}{T_{enc}} \quad (3.3.1)$$

$$Throughput_{dec} = \frac{\text{Ukuran data}}{T_{dec}} \quad (3.3.2)$$

4) *Avalanche effect*

Avalanche effect digunakan untuk mengukur perubahan bit pada *ciphertext* akibat perubahan satu bit pada plaintext. Nilai *Avalanche effect* yang baik menunjukkan bahwa perubahan kecil pada input mampu menghasilkan perubahan signifikan pada output. Dalam konteks penelitian ini, *Avalanche effect* dihitung menggunakan formula (3.4).

$$AE = \frac{Jumlah\ bit\ yang\ berubah}{Total\ bit\ ciphertext} x100\% \quad (3.4)$$

5) Entropi Shannon

Entropi Shannon digunakan untuk mengukur tingkat keacakan distribusi byte pada *ciphertext*. Nilai entropi yang mendekati 8 menunjukkan bahwa distribusi byte pada *ciphertext* semakin acak, sehingga pola statistik dari

plaintext semakin sulit dikenali. Formula entropi Shannon untuk variabel diskrit X didefinisikan pada formula (3.5).

$$H(X) = -\Sigma\, p(x_i) \times \log 2\, p(x_i) \quad (3.5)$$

$p(x_i)$ adalah probabilitas kemunculan nilai ke-i. Untuk data 8-bit, nilai entropi maksimum adalah 8,0 bit yang tercapai ketika setiap nilai byte (0–255) muncul dengan probabilitas sama yaitu 1/256.

Entropi Shannon yang mendekati nilai maksimum pada *ciphertext* mengindikasikan bahwa algoritma berhasil mengaburkan pola statistik dari *plaintext* asli, sebuah manifestasi dari properti confusion yang baik. Sebaliknya, *ciphertext* dengan nilai entropi yang jauh di bawah 8,0 mengindikasikan adanya distribusi byte yang tidak seragam, yang berpotensi memberikan informasi parsial kepada penyerang (Zolfaghari dkk., 2022).

6) *Key sensitivity*

Algoritma enkripsi yang memiliki *Key sensitivity* tinggi akan menghasilkan perubahan *ciphertext* yang besar akibat perubahan kecil pada kunci, sehingga kualitas difusi algoritma meningkat. Oleh karena itu, semakin mendekati atau melebihi 50% perubahan bit *ciphertext* akibat perubahan satu bit kunci, maka kualitas difusi *ciphertext* dianggap semakin baik (Patidar & Kaur, 2023). Formula *Key sensitivity* dapat dituliskan pada Persamaan (3.6).

$$KS = \frac{\text{Jumlah bit } \textit{ciphertext} \text{ yang berubah akibat perubahan kunci}}{\text{Total bit } \textit{ciphertext}} \times 100\% \quad (3.6)$$

7) *Frequency test*

Dalam penelitian ini, *Frequency test* digunakan untuk mengevaluasi tingkat keacakan *ciphertext* yang dihasilkan oleh algoritma AES-128 termodifikasi. Semakin kecil selisih antara jumlah bit 0 dan bit 1, semakin baik kualitas randomisasi *ciphertext* yang dihasilkan. Formula *Frequency test* dapat dituliskan pada persamaan (3.7).

$$F = \frac{\mathrm{n}_1 - \mathrm{n}_0}{n} \tag{3.7}$$

8) *Runs test*

Runs test pada NIST menggunakan proporsi bit 1 pada sequence, jumlah run aktual, dan p-value Semakin acak urutan bit, pola pergantiannya tidak terlalu teratur. NIST mendefinisikan *Runs test* berdasarkan total run, yaitu rangkaian bit yang sama secara berurutan. Formula *Runs test* dapat dituliskan pada persamaan (3.8).

$$R = \sum_{i=1}^{n-1} d_i \tag{3.8}$$

9) *Chi-Square test*

Pada penelitian ini, *Chi-Square test* digunakan untuk mengevaluasi tingkat keseragaman distribusi *ciphertext* sebagai salah satu indikator bahwa proses enkripsi berhasil menyamarkan karakteristik plaintext secara efektif.

$$\chi^2 = \sum_{i=1}^{k} \frac{(\mathbf{Oi} - \mathbf{Ei})^2}{E_i} \tag{3.9}$$

3.5.5. Penarikan Kesimpulan

Tahapan terakhir adalah penarikan kesimpulan berdasarkan hasil analisis dan perbandingan yang telah dilakukan pada tahap sebelumnya. Kesimpulan

dirumuskan secara sistematis dengan menjawab setiap butir rumusan masalah yang telah ditetapkan pada Bab I, mencakup keberhasilan perancangan mekanisme *dynamic bit permutation* berbasis *logistic chaotic map*, serta perbandingan performa algoritma yang diusulkan dengan penelitian baseline dari Kecepatan Enkripsi, Kecepatan Dekripsi, *Avalanche effect*, *Throughput* , Entropi Shannon, *Key sensitivity*, *Frequency test*, *Runs test*, dan *Chi-Square test*. Selain kesimpulan, dirumuskan pula saran untuk penelitian lanjutan berdasarkan keterbatasan penelitian ini.