# BAB II
# DASAR TEORI

## 2.1 KAJIAN PUSTAKA

Terdapat penelitian mengenai kriptografi *serpent* dengan *twofish* yang dilakukan oleh Sutan, dkk [9]. Pada tahun 2020, sebuah penelitian berjudul "Analisis Perbandingan Kinerja Algoritma Kriptografi *Serpent* dan *Twofish* pada *Dataset World Bank Projects and Operations*" dilakukan. Tujuan dari penelitian tersebut adalah untuk menilai keunggulan salah satu algoritma kriptografi dalam proses enkripsi dan dekripsi berkas. Dalam eksperimen awal menggunakan kunci 128 bit, *twofish* menunjukkan kecepatan enkripsi dan dekripsi rata-rata sebesar 43% dan 35%, lebih cepat daripada *serpent*. Dengan menggunakan kunci 192-bit, kecepatan enkripsi dan dekripsi algoritma *twofish* meningkat sebesar 40% dan 48% dibandingkan dengan *serpent*. Meskipun pada kunci 256-bit, kecepatan enkripsi *twofish* meningkat 42% dan 48% dibandingkan dengan *serpent*, namun pada saat dekripsi *file* oleh *twofish* memakan waktu yang cukup lama. Penurunan kecepatan ini disebabkan oleh kompleksitas kombinasi *key* yang sangat besar, bahkan kunci yang berukuran 128-bit. Secara keseluruhan, *twofish* dianggap lebih unggul dibandingkan dengan *serpent*.

Kemudian, terdapat penelitian mengenai *blowfish* dengan *twofish* yang dilakukan oleh Muhathir [10] pada tahun 2018 dengan judul “Perbandingan Algoritma *Blowfish* dan *Twofish* Untuk Kriptografi *File* Gambar”. Dalam penelitian ini, dilakukan eksperimen untuk menguji tingkat kecepatan algoritma *blowfish* dan *twofish* dalam melaksanakan proses enkripsi dan dekripsi *file* gambar. Pengujian tersebut dilakukan melalui penerapan aplikasi yang mampu melakukan enkripsi pada *file* gambar dan mencatat hasil kecepatan proses yang dibutuhkan oleh *blowfish* dan *twofish.* Dari gambar yang diberikan, salah satu contohnya adalah berukuran 1200×900, gambar tersebut dienkripsi dan didekripsi menggunakan *blowfish* dan *twofish*, untuk hasilnya adalah Ketika algoritma *twofish* ditugaskan untuk mengenkripsi, diperlukan waktu hingga 892 ms sedangkan untuk dekripsi memerlukan waktu 942 ms, hasil ini lebih cepat dibandingkan *blowfish* yang perlu waktu untuk enkripsi dan dekripsi selama 923 ms dan 976 ms. Terdapat banyak pengujian yang Muhathir lakukan, namun secara keseluruhan percobaan, algoritma

*twofish* memiliki waktu yang lebih cepat dibandingkan *blowfish,* maka dari itu, dapat diberi kesimpulan bahwa *twofish* lebih unggul dalam segi kecepatan untuk mengenkripsi dan mendekripsi suatu *file* gambar dibandingkan dengan *blowfish*. Rerata perbandingan yang dihasilkan dari keduanya dalam satuan milidetik adalah 4355:4267. Faktor kecepatan ini juga bergantung pada ukuran *file*; semakin besar *file* yang diolah oleh *twofish* dan *blowfish*, semakin lama waktu yang dibutuhkan untuk melaksanakan proses enkripsi dan dekripsi.

Terdapat penelitian mengenai *twofish* oleh Siswanto, dkk [11]. pada 2021 dengan topik "Penerapan Algoritma Kriptografi *Twofish* Untuk Mengamankan Data *File*". Dalam rangka penelitian ini, algoritma kriptografi *twofish* diaplikasikan untuk menjamin keamanan data *file* dalam suatu aplikasi. Dalam percobaan yang dilaksanakan, 15 *file* telah dienkripsi, dan hasilnya menunjukkan peningkatan rata-rata ukuran *file* sebesar 0,11%, dengan waktu enkripsi sekitar 9,5352 ms dan waktu dekripsi sekitar 9,3294 ms. Pengujian UAT dilakukan menggunakan kuesioner dengan skala Likert berjumlah 5 poin. Temuannya menunjukkan bahwa lebih dari 91,23% responden setuju bahwa implementasi aplikasi dengan algoritma twofish dapat menjaga kerahasiaan data *file* secara keseluruhan.

Kemudian, terdapat penelitian mengenai *twofish* yang dilakukan oleh Laylim, dkk [12]. pada tahun 2019, dilakukan sebuah penelitian berjudul "Penerapan Algoritma *Twofish* dalam Aplikasi *Chat* Android". Penelitian ini difokuskan pada pemanfaatan kriptografi *twofish* untuk aplikasi pesan teks berbasis Android, dengan menggunakan algoritma tersebut untuk melindungi pertukaran pesan teks melalui proses enkripsi dan dekripsi. Penelitian oleh Florensius dan timnya menyimpulkan bahwa *twofish* efektif untuk amankan pesan teks pada aplikasi *chatting*, berkat kecepatan relatif cepat dalam proses enkripsi dan dekripsi. Dengan demikian, *twofish* bisa menjadi pilihan terbaik untuk menjaga data.

Setelah meninjau beberapa daftar pustaka dari penelitian yang terdahulu mengenai algoritma kriptografi, berikut tabel rangkuman dari referensi yang digunakan pada daftar pustaka kali ini, seperti terlihat pada tabel 2.1.

**Tabel 2.1 Ringkasan terkait kajian Pustaka**

| Peneliti | Judul | Komponen Penelitian | | | |
|---|---|---|---|---|---|
| | | Jenis Kriptografi | Parameter Pengujian | Tujuan | Hasil |
| Rio Sudrajat Puji Sutan, dkk. | Analisis Perbandingan Kinerja Algoritma Kriptografi *Serpent* dan *Twofish* pada *Dataset "World Bank Projects and Operations"* | *Serpent* dan *Twofish* | Kecepatan enkripsi-dekripsi | Mengetahui perbandingan kedua algoritma dari segi kecepatan | *twofish* lebih unggul 35% dan 43% dalam segi kecepatan enkripsi & dekripsi dibandingkan *serpent* |
| Muhathir | Perbandingan Algoritma *Blowfish* dan *Twofish* untuk Kriptografi File Gambar | *Blowfish* dan *Twofish* | Kecepatan enkripsi-dekripsi *file* gambar | Mengetahui perbandingan kecepatan antar dua algoritma | *Twofish* lebih cepat dalam melakukan enkripsi-dekripsi dibandingkan *blowfish* |
| Siswanto, dkk. | Penerapan Algoritma Kriptografi *Twofish* Untuk Mengamankan Data File | *Twofish* | Implementasi *twofish* ke aplikasi enkripsi-dekripsi *file* | Mengimplementasikan *twofish* ke dalam aplikasi untuk mengenkripsi-dekripsi *file* | 15 *file* telah dienkripsi dengan ukuran *file* meningkat sebesar 0,11% dengan waktu proses 9,53ms |
| Florensius Laylim, dkk. | Penerapan Algoritma *Twofish* Dalam Perancangan Aplikasi *Chat* Berbasis Android | *Twofish* | Implementasi twofish ke aplikasi *chat* enkripsi-dekripsi pesan teks | Mengimplementasikan *twofish* ke dalam aplikasi *chat* untuk enkripsi-dekripsi pesan teks | Aplikasi *chatting* cukup cocok digunakan dengan mengimplementasi algoritma kriptografi *twofish* |
| Maharani Aditya Putri | Analisis Performansi Algoritma Kriptografi *Twofish* pada Proses Enkripsi-Dekripsi Pesan Teks dan Gambar | *Twofish* | Analisis performansi *twofish* untuk mengenkripsi dan dekrpisi pesan teks dan gambar | Mengetahui hasil kinerja *twofish* berdasarkan kecepatan proses enkripsi dan dekripsi pesan teks dan gambar | *Twofish* dapat digunakan untuk mengenkripsi dan dekripsi teks dan *file* gambar, dengan waktu yang relatif cepats |

Dari tinjauan Pustaka tersebut, akan dilakukan sebuah penelitian dengan memakai kriptografi *twofish* karena dari segi kecepatan dan keamanan, kriptografi tersebut sudah dipercaya memiliki kinerja yang baik, oleh karena itu, pada penelitian kali ini akan dilakukan penelitian mengenai kriptografi *twofish* yang akan dimanfaatkan untuk mengenkripsi dan dekripsi sebuah data dalam bentuk pesan teks dengan menggunakan program Python sederhana di sebuah komputer dengan sistem operasi Windows.

## 2.2 DASAR TEORI

Pada Dasar Teori ini akan membahas mengenai teori yang relevan di dalam penelitian, yaitu antara lain:

### 2.2.1 Keamanan Data

Keamanan data informasi melibatkan tindakan-tindakan yang diambil untuk melindungi data dari potensi kerusakan atau penyalahgunaan oleh individu, baik dari dalam maupun luar suatu organisasi. Dalam sistem informasi, teknologi informasi digunakan untuk menjaga keamanan data dengan menggabungkan teknologi keamanan telekomunikasi dan teknologi keamanan komputer. Tingkat keamanan data juga tergantung pada tingkat sensitivitas data yang tersimpan dalam database. Seorang *administrator* data perlu aktif dan proaktif dalam mengimplementasikan langkah-langkah keamanan guna melindungi integritas informasi data. Data *digital* memiliki nilai yang setara dengan informasi fisik dan oleh karena itu perlu dijaga dengan keunggulan dan kerahasiaan yang seimbang. Beberapa teknik yang digunakan dalam menjaga keamanan data mencakup kriptografi dan steganografi [13].

### 2.2.2 Algoritma Kriptografi

Kriptografi merupakan elemen vital dalam keamanan siber yang berperan signifikan dalam menjaga kerahasiaan data dan jalur komunikasi. Pada dasarnya, kriptografi adalah suatu Teknik yang erat hubungannya dengan enkripsi, yaitu proses mengubah data menjadi format atau kode tertentu. Untuk membaca informasi tersebut, diperlukan kunci atau *key* untuk proses dekripsi, yaitu proses mengubah data yang telah dienkripsi agar Kembali ke bentuk aslinya. Selama

informasi ditransmisikan atau disimpan, kerahasiaan data pesan tetap terjaga, sehingga enkripsi membantu memastikan kerahasiaan menjadi prioritas utama [14].

Kriptografi sendiri memiliki beberapa macam tujuan, setidaknya terdapat empat tujuan utama kriptografi, yaitu:

1. Kerahasiaan

   Menjaga data pribadi atau perusahaan, sehingga informasi yang ada, hanya untuk pihak yang memiliki akses saja.

2. Integritas

   Membuat informasi yang tersedia, tidak bisa dimanipulasi oleh siapapun.

3. Autentikasi

   Dapat memastikan pihak yang mengakses informasi dan mengonfirmasi keaslian dari identitas pengguna beserta informasinya.

4. Antipenyangkalan

   Menghindari situasi di mana pengguna menolak untuk mematuhi atau melaksanakan komitmen atau tindakan yang telah dilakukan sebelumnya.

Kriptografi juga memiliki berbagai jenis yang berbeda, diantaranya adalah sebagai berikut:

1. *Public key cryptography*

   Kriptografi kunci public adalah sebuah proses yang dianggap lebih aman dan terjamin, menggunakan dua kunci yang saling terkait, yaitu kunci privat dan kunci publik.

   Meskipun kunci publik dapat diberikan secara bebas, namun ketika dikombinasikan dengan kunci privat, kode enkripsi dan data yang terkandung di dalamnya tetap terjaga kerahasiannya. Beberapa contoh metode kriptografi yang sering digunakan dalam hal ini antara lain adalah DSA, RSA, PKC dan *elliptic curve techniques.*

2. *Symetric key cryptography*

   Salah satu jenis kriptografi lainnya adalah *symmetric key cryptography* atau yang dalam bahasa Indonesia disebut sebagai kriptografi kunci simetris. Pada jenis kriptografi ini, pengirim dan

penerima pesan atau data menggunakan hanya satu kunci untuk melakukan enkripsi dan dekripsi informasi.

3. *Hash function*

   Fungsi *hash* adalah sebuah algoritma enkripsi satu arah yang tidak melibatkan penggunaan kunci untuk melindungi data. Fungsi *hash* secara mendasar adalah suatu fungsi matematika yang mengubah *input* berupa nilai numerik menjadi pesan terkompresi [15].

### 2.2.3 Algoritma *Twofish*

*Twofish* adalah sebuah algoritma kriptografi kunci simetris berbasis blok dengan panjang blok tetap 128 bit. Sebagai salah satu algoritma kriptografi simetris, algoritma *twofish* memiliki Panjang blok tetap 128-bit dan variasi panjang kunci, mulai dari 128, 192, hingga 256-bit. Algoritma *twofish* menggunakan beberapa Teknik seperti *whitening*, *Feistel Network*, manipulasi bit, Transformasi Pseudo-Hadamard, ekspansi *filter*, pemutaran kunci hingga 16 kali dan *Most Distance Separable* atau MDS[8].

*Twofish* juga adalah sebuah algoritma yang kuat dan hingga saat ini dianggap aman karena belum ada kriptanalisis yang berhasil meretasnya. Selain itu, algoritma ini tidak dipatenkan, sehingga penggunaan dan implementasinya dalam perangkat enkripsi tidak memerlukan biaya tambahan.

Algoritma *twofish* adalah salah satu algoritma yang disarankan sebagai pengganti AES. Keberhasilkan *twofish* memenuhi persyaratan desain yang ditetapkan NIST (*National Institute of Standards and Technology*) sebagai standar AES dapat dijelaskan dengan beberapa faktor berikut:

a. Blok *chiper* simetris 128bit.
b. Panjang kunci yang digunakan adalah 128bit, 192bit, dan 256bit.
c. Kunci-kunci yang digunakan tidak ada yang lemah.
d. *Platform* yang tersedia untuk *twofish* memiliki Tingkat efisiensi pada *software* dan *hardware*.
e. *Twofish* menunjukkan tingkat fleksibilitas yang tinggi dalam penggunaannya, sesuai untuk *stream cipher*, fungsi *hash*, dan MAC. Sebagai contoh, dengan kemampuan menerima panjang kunci

tambahan, algoritma ini dapat diimplementasikan pada perangkat lunak dan perangkat keras dengan berbagai *platform*.

f. Desain dari *twofish* cukup simpel, dapat memudahkan untuk analisa atau implementasi.

Dari standar AES tersebut, didapatkan beberapa keunggulan algoritma *twofish,* yaitu:

1. Pada setiap *round,* memiliki varian dengan sebuah nomor variabel.
2. *Twofish* menawarkan *key schedule* yang dapat diprakomputasikan, yang memungkinkan untuk mencapai kecepatan maksimum dalam pengolahan data dan penggunaan memori yang efisien. *Key schedule* ini dirancang sedemikian rupa sehingga sebagian besar komputasi kunci dapat dilakukan sebelum proses enkripsi atau dekripsi dimulai. Dengan demikian, algoritma ini dapat mengoptimalkan waktu eksekusi dan meminimalkan penggunaan memori yang dibutuhkan selama proses kriptografi. Keunggulan ini membuat *twofish* menjadi pilihan yang baik untuk situasi di mana kecepatan dan penggunaan memori yang efisien menjadi faktor kunci.
3. *Twofish* dapat diandalkan sebagai *stream chipper, one-way hash function,* MAC dan *pseudo-random number generator*. Algoritma ini dirancang dengan metode konstruksi yang mudah dipahami, sehingga dapat digunakan dengan baik dalam berbagai aplikasi keamanan. *Twofish* memberikan kinerja yang optimal dan keandalan yang tinggi dalam menghasilkan aliran data acak, menghitung *hash* nilai satu arah, dan menghasilkan kode otentikasi pesan. Dalam penggunaannya, *twofish* dapat dengan mudah diimplementasikan dan dimengerti oleh pengembang yang berbeda, menjadikannya pilihan yang sesuai untuk kebutuhan kriptografi yang beragam.
4. *Twofish* menyediakan variasi *famili-key* yang memungkinkan adanya versi *chiper* yang berbeda-beda dan tidak dapat diinterupsi. Dengan varian *famili-key* ini, *twofish* dapat digunakan dalam berbagai implementasi *chiper* yang berbeda, yang menawarkan fleksibilitas dalam pemilihan algoritma dan kunci yang sesuai dengan kebutuhan

spesifik. Selain itu, kemampuan *twofish* untuk menjadi *non-interruptible*, atau tidak dapat diinterupsi, menjadikannya lebih tahan terhadap serangan dan manipulasi oleh pihak yang tidak berwenang. Hal ini memberikan tingkat keamanan yang lebih tinggi dan menjaga integritas data saat digunakan dalam lingkungan yang rentan terhadap gangguan atau serangan [16].

Algoritma *twofish* juga memiliki kelemahan, yaitu memerlukan kunci yang berbeda untuk setiap pasangan pengguna. Namun, tantangan muncul ketika sulit untuk menyimpan dan mengingat banyak kunci dengan nama pengguna yang berbeda. Oleh karena itu, manajemen kunci menjadi suatu tugas yang sulit. Perlu ada kesepakatan mengenai jalur khusus untuk kunci, namun hal ini juga dapat menimbulkan kendala baru karena sulit menentukan jalur yang aman untuk menyimpan kunci. Kendala ini sering dikenal sebagai "Masalah Distribusi Kunci". Jika kunci terhilang atau dapat ditebak, keamanan sistem kriptografi menjadi terancam [17].

### 2.2.4 Blok Pembangun *Twofish*

Pada algoritma *twofish* memiliki beberapa blok dalam membangun sebuah algoritma *twofish,* pada *twofish* sendiri menggunakan suatu *block chiper* yang dapat digunakan untuk menciptakan aliran *chiper* dengan berbagai metode sinkronisasi untuk mencapai berbagai properti kesalahan. Selain itu, *block chiper* juga dapat digunakan sebagai fungsi *hash,* pengkodean pengesahan pesan, dan pembangkit bilangan *pseudo-random* [18]. Berikut contoh blok yang membangun algoritma *twofish* seperti di bawah ini:

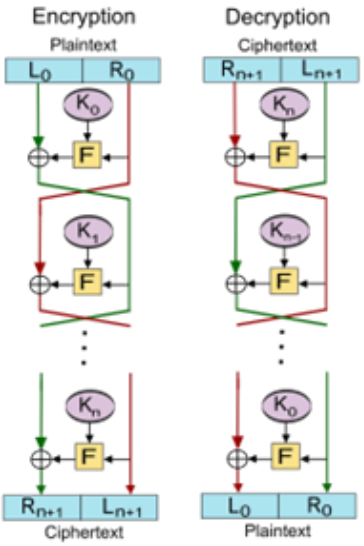
#### 2.2.4.1 Jaringan Feistel

Jaringan Feistel yang merupakan struktur simetris yang digunakan untuk membangun *block chiper,* ditemukan oleh Horst Feistel pada tahun 1970. Sejak penemuan itu, jaringan Feistel telah menjadi komponen umum dalam banyak algoritma enkripsi seperti DES, GOST, Lucifer, Triple DES, dan banyak lainnya [19].

Jaringan Feistel menggunakan fungsi ronde yang berperan dalam mengolah dua masukan, yaitu blok data dan subkunci. Fungsi ini menghasilkan keluaran dengan ukuran yang identik dengan blok data. Pada setiap tahap ronde, fungsi ronde

digunakan untuk memproses separuh data yang akan dienkripsi, lalu hasilnya di-XOR-kan dengan separuh data yang lain. Proses ini diulang beberapa kali dengan jumlah yang konstan. Pada akhirnya, data akan berhasil dienkripsi setelah melewati serangkaian langkah tersebut.

Peneliti Michael Luby dan Charles Rackoff [20] Melakukan analisis terhadap struktur sandi Feistel dan mengonfirmasi bahwa jika fungsi ronde yang digunakan merupakan fungsi acak semu yang aman secara kriptografi dengan menggunakan kunci Ki, maka tiga tahap ronde sudah mencukupi untuk menghasilkan penyandian blok yang bersifat permutasi acak semu. Dengan penambahan satu tahap ronde menjadi empat, permutasi akan lebih "kuat", sehingga permutasi tersebut akan tetap bersifat acak semu bahkan jika seseorang memiliki akses ke inversi permutasinya. Karena hasil penelitian ini memiliki signifikansi yang besar, sandi Feistel kadang-kadang dikenal sebagai penyandian blok Luby-Rackoff. Terdapat juga contoh sederhana dari jaringan Feistel seperti yang diperlihatkan pada gambar 2.1 [21].




**Gambar 2.1 Jaringan Feistel sederhana [21]**

Pada gambar 2.1 menunjukkan sebuah jaringan Feistel enkripsi dan dekripsi yang memiliki fungsi untuk mengenkripsi *plaintext* dan juga mendekripsi *chipertext*. Jaringan Feistel akan membagi blok *plaintext* menjadi berukuran n bit dengan dua bagian, kiri (L) dan kanan (R). Kemudian, pada gambar terdapat F dan K, misalkan F sebagai fungsi ronde, sedangkan K sebagai subkunci untuk ronde ke-

0, 1, ….., n. Kemudian, setelah melakukan n putaran ketika proses enkripsi, karena blok *plaintext* adalah ($L_0$, $R_0$), maka *block chiper*-nya adalah ($R_{n+1}$, $L_{n+1}$), begitupun sebaliknya dari *block chiper* ke *plaintext* Ketika dilakukan proses dekripsi [21].

Secara singkat dari gambar 2.1, struktur dasar atau model dasar dari jaringan Feistel adalah sebagai berikut:

a. Proses enkripsi
    1. *Plaintext* yang dimasukkan akan dibagi menjadi dua blok teks yang sama besar, yaitu $L_{0,}$ $R_0$.
    2. Pada setiap ronde yang dilakukan, misalkan ke-i = 0, 1, …., n, maka

$$L_{i+1} + R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_I) \text{ ……(2.1)}$$

    Dengan $\oplus$ adalah operasi XOR.
    3. Dari hasil putaran tersebut adalah teks terenkripsi ($R_{n+1}$, $L_{n+1}$).

b. Proses dekripsi
    1. *Chipertext* yang dimasukkan akan dibagi menjadi dua blok teks yang sama besar, yaitu $R_{n+1}$ dan $L_{n+1}$.
    2. Pada setiap ronde yang dilakukan, misalkan ke-i = n, n-1, …., 0, maka

$$R_i = L_{i+1}$$

$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i) \text{ ……(2.2)}$$

    3. Dari hasil putaran tersebut adalah teks asli ($L_0$, $R_0$).

Pada proses enkripsi terdapat proses yang terjadi saat putaran enkripsi berlangsung, hal itu dapat diketahui dengan persamaan 2.1, diketahui pada saat proses enkripsi akan dilakukan, *plaintext* akan dibagi menjadi dua bagian yang sama besar yaitu $L_0$ (untuk blok kiri) dan $R_0$ (untuk blok kanan), pada putaran ke i+1 untuk blok teks sebelah kiri, akan sama dengan blok teks kanan ke-i, kemudian ketika blok teks R sudah melakukan putaran ke i+1, akan sama dengan blok teks L ke-i, yang sudah dilakukan operasi XOR dengan F (fungsi) dan blok teks R ke-i beserta dengan subkunci K ke-i. Maka, hasil dari persamaan tersebut, akan menjadi teks yang sudah terenkripsi.

Kemudian, setelah proses enkripsi, akan dilakukan proses dekripsi agar pesan teks asli terlihat, dengan menggunakan persamaan 2.2, diketahui pada saat proses dekripsi akan dilakukan, *chipertext* akan dibagi menjadi dua bagian yang sama besar yaitu $R_{n+1}$ (untuk blok kiri) dan $L_{n+1}$ (untuk blok kanan), pada putaran ke-i untuk blok teks R akan sama dengan blok teks L ke i+1, kemudian ketika blok teks L ke-i akan sama dengan blok teks R ke i+1, yang sudah dilakukan operasi XOR dengan F (fungsi) dan blok teks L ke i+1 beserta dengan subkunci K ke-i. Maka, hasil dari persamaan tersebut akan menjadi teks yang sudah terdekripsi [21].

Untuk mempermudah proses tersebut, dalam satu putaran ke-i jaringan Feistel adalah:

$$X_{i+1} = \left( F_{ki} \left( msb_{\frac{n}{2}}(X_i) \right) \oplus \ lsb_{\frac{n}{2}}(X_i) \right) \text{.....(2.3)}$$

Persamaan 2.3 adalah rumus sederhana jaringan Feistel, pada rumus tersebut terdapat blok keluaran $X_{i+1}$ (blok teks $R_0$ dan $L_0$) yang sama dengan F (fungsi) subkunci K ke-i yang dikalikan sebelumnya dengan $msb_{n/2}$ atau n/2 dari *most dignificant bit* dari blok masukan $X_i$. Setelah itu, dari F tersebut akan dioperasikan XOR dengan $lsb_{n/2}$ atau n/2 dari *least significant bit* dari blok masukan $X_i$ [19].

### 2.2.4.2 Kotak-S (*S-Boxes*)

Kotak-S adalah matriks yang berfungsi sebagai penggantian substansi non-linier, me-*remapping* satu atau beberapa bit menjadi satu atau beberapa bit lainnya, dan digunakan dalam berbagai blok kode. Dimensi masukan dan keluaran Kotak-S dapat bervariasi. Terdapat empat pendekatan yang diterapkan untuk mengisi nilai Kotak-S, yakni dengan pemilihan acak, pemilihan acak yang diikuti pengujian, pembuatan manual oleh manusia, dan perhitungan matematis. Kotak-S pertama kali diperkenalkan dalam algoritma Lucifer, selanjutnya diadopsi oleh DES, dan menjadi fitur umum dalam berbagai algoritma enkripsi lainnya. Dalam konteks twofish, empat blok-S berukuran 8x8 bit yang berbeda, bersifat bijektif, dan bergantung pada kunci, telah diciptakan dengan menggunakan permutasi 8x8 bit serta materi kunci [18]. Untuk tabel *S-Boxes* seperti terlihat pada gambar 2.2

| C (mn) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | FA | A6 | A9 | E5 | DE | 5A | 05 | FB | 5C | AA | 64 | CB | A1 | 87 | 6A | 6C |
| 1 | E4 | 87 | 93 | 6A | 76 | E5 | 66 | 09 | 43 | 0C | 91 | 92 | 03 | 8F | 63 | 30 |
| 2 | 54 | 99 | E9 | 30 | EE | BF | F2 | E6 | 71 | 4E | 90 | D5 | 18 | 85 | 45 | FA |
| 3 | 7E | 73 | 0E | 13 | 8B | 5B | 08 | 8B | C8 | 3B | 6A | 10 | 87 | 09 | FB | 47 |
| 4 | 28 | AF | C5 | 20 | 0B | 8D | 74 | D5 | 59 | 37 | 19 | C9 | 2A | 4F | 02 | C1 |
| 5 | 91 | F1 | 50 | 83 | 9B | 42 | 87 | 4A | 42 | F2 | 74 | 0C | 4F | 2D | 49 | AE |
| 6 | DA | 25 | 64 | 58 | CD | FE | 1B | D2 | 7D | F8 | 66 | A8 | 6D | 2A | A9 | 7E |
| 7 | 21 | C3 | 3F | D2 | EC | C9 | 7A | AC | 0D | CC | 7F | D3 | 35 | 25 | C2 | 6F |
| 8 | BF | 86 | 07 | 91 | 5E | C5 | 75 | 4E | C1 | 83 | E8 | CA | B0 | E9 | 75 | B6 |
| 9 | 07 | 16 | F8 | 7B | 49 | D5 | FA | AD | 5B | 5F | C2 | 19 | 12 | 13 | C5 | 20 |
| A | 99 | B3 | 44 | 9F | F2 | 71 | 9B | 38 | 7A | AE | A5 | 0D | 48 | C4 | EF | 1E |
| B | F4 | 09 | 8F | D1 | 2F | 88 | 60 | 9B | E9 | 9F | 75 | 66 | 69 | 7C | 23 | 62 |
| C | 05 | DE | 30 | DE | BB | D5 | 96 | 4D | 52 | AB | 77 | 32 | 09 | B4 | 3F | AB |
| D | FF | 97 | D6 | FB | FE | 59 | DB | BA | 15 | 1A | 64 | 2D | 02 | F8 | 5D | 3B |
| E | 74 | EE | 1B | 82 | C8 | 63 | F8 | F4 | BF | 49 | 50 | 5A | 7C | FC | 46 | 47 |
| F | C9 | 97 | ED | 52 | 59 | D3 | 01 | 56 | 79 | DB | C7 | 9A | E7 | 26 | 12 | 00 |

**Gambar 2.2 Tabel *S-Boxes***

#### 2.2.4.3 MDS *Matrices*

Kode *Maximum Distance Separable* (MDS) pada suatu *field* merujuk pada suatu pemetaan linear dari x elemen *field* ke y elemen field, yang menghasilkan vektor komposit dengan total elemen x + y. Dalam pemetaan ini, terdapat syarat bahwa jumlah minimum elemen non-nol pada setiap vektor non-nol harus paling tidak y + 1. Dengan kata lain, jumlah elemen yang berbeda antara dua vektor yang berbeda yang dihasilkan oleh pemetaan MDS minimal adalah y + 1. Secara mudah dapat dibuktikan bahwa tidak ada pemetaan yang dapat memiliki jarak terbesar antara dua vektor yang berbeda, sehingga konsep ini dikenal sebagai jarak pisah maksimum (*maximum distance separable*). Representasi dari pemetaan MDS dapat dilakukan menggunakan sebuah matriks MDS berukuran x × y elemen. *Reed-Solomon* (RS) *code*, yang umumnya digunakan untuk perbaikan kesalahan, adalah contoh konkret dari kode MDS. Syarat untuk memastikan suatu matriks berukuran x × y dapat dianggap sebagai MDS adalah setiap submatriks kotak yang mungkin, yang diperoleh dengan menghapus kolom atau baris, harus memiliki determinan yang tidak nol. Serge Vaudenay adalah orang pertama yang memperkenalkan matriks MDS sebagai bagian dari desain kode. Kode-kode seperti *Shark* dan *Square* juga memanfaatkan matriks MDS, meskipun konsep konstruksinya pertama kali ditemukan dalam kode Manta yang tidak pernah dipublikasikan. *Twofish* sendiri menggunakan satu matriks MDS dengan ukuran 4x4[8].

#### 2.2.4.4 Transformasi Pseudo-Hadamard (PHT)

*Pseudo-Hadamard Transform* (PHT) adalah suatu operasi pencampuran yang simpel dan efisien yang bekerja dengan cepat pada perangkat lunak berbasis 32-bit. Operasi ini melibatkan dua masukan yang telah ditentukan:

$$a` = a + b \; mod \; 2^{32} \; \ldots\ldots \; (2.4)$$

$$b` = a + 2b \; mod \; 2^{32} \; \ldots.. \; (2.5)$$

Dalam persamaan 2.4 dan 2.5, terdapat operasi dua masukan untuk Transformasi Pseudo-Hadamard yang diwakili oleh a **dan b** dengan basis 32-bit. Algoritma *twofish* menggunakan Transformasi Pseudo-Hadamard berbasis 32-bit dalam mengubah keluaran dari fungsi G, yang merupakan komponen inti dari seluruh algoritma *twofish*. Masukan X sebesar 32-bit dari fungsi F dibagi menjadi empat bagian, masing-masing berukuran 8-bit. Setiap bagian 8-bit kemudian diproses menggunakan kotak S yang sesuai. Setiap kotak S berperan sebagai fungsi bijektif yang menerima 8-bit sebagai masukan dan menghasilkan 8-bit sebagai keluaran. Keempat keluaran 8-bit ini kemudian dikalikan dengan matriks *Most Distance Separable* (MDS) berukuran 4x4. Hasil perkalian ini kemudian diinterpretasikan sebagai 32-bit, menjadi keluaran dari fungsi G, dan akan dikembalikan ke fungsi F [22].

#### 2.2.4.5 *Whitening*

*Whitening* adalah teknik yang digunakan untuk meningkatkan tingkat keamanan pada *cipher block* yang melibatkan serangkaian iterasi. Tujuan dari *whitening* adalah untuk menjaga kerahasiaan *input* dan *output* dari fungsi F. Proses *whitening* melibatkan operasi XOR antara data dan bagian dari kunci sebelum iterasi pertama, serta setelah iterasi terakhir dalam proses enkripsi [23].

Sebelum putaran awal algoritma Feistel, *twofish* menjalankan operasi XOR antara subkunci 128-bit dan *plaintext*, dan mengulangi operasi XOR ini setelah putaran terakhir dari algoritma Feistel. Subkunci ini dihasilkan dengan proses yang serupa seperti pembentukan subkunci pada setiap putaran, walaupun tidak dimanfaatkan dalam bagian lain dari implementasi *twofish* [18].

#### 2.2.4.6 Penjadwalan Kunci

Penjadwalan kunci dalam algoritma *twofish* melibatkan transformasi bit-bit kunci menjadi upa-kunci, yaitu hasil dari proses penjadwalan kunci dalam kriptografi. Upa-kunci tersebut kemudian dapat digunakan oleh kode untuk melakukan operasi enkripsi dan dekripsi. Dengan melakukan penjadwalan kunci sebelumnya, seluruh upa-kunci dapat dihasilkan dan disimpan, sehingga saat proses enkripsi atau dekripsi dilakukan, kode hanyatperlu mengakses upa-kunci yang telah disiapkan sebelumnya. Hal ini mengoptimalkan kecepatan proses, karena tidak perlu melakukan transformasi kunci setiap kali operasi enkripsi atau dekripsi dilakukan, melainkan dapat langsung menggunakan upa-kunci yang telah di-*generate* sebelumnya [18].

Dalam mengimplementasikan algoritma *twofish*, perlu dipertimbangkan kecepatan komputasi yang diinginkan. *Twofish* memiliki karakteristik tertentu yang melibatkan proses persiapan kunci yang memakan waktu yang cukup lama. Untuk mengoptimalkan kecepatan, ada pendekatan yang dapat dilakukan yaitu melakukan semua proses penjadwalan kunci sebelumnya dan menyimpannya. Dengan demikian, tidak perlu melakukan penjadwalan kunci setiap kali proses enkripsi atau dekripsi dilakukan. Dengan melakukan penjadwalan kunci pada tahap awal, implementasi *Twofish* dapat menyelesaikan proses enkripsi dan dekripsi secara cepat dan efisien.

Tetapi, jika implementasi algoritma memperhatikan penggunaan memori yang minimal, *twofish* juga mampu melakukan penjadwalan kunci secara simultan dengan proses enkripsi dan dekripsi. Dalam pendekatan ini, penjadwalan kunci dilakukan secara bertahap seiring dengan perkembangan proses enkripsi dan dekripsi, sehingga penggunaan memori dapat diminimalkan [24].

Dalam setiap iterasi 16 putaran, dua kata pertama berperan sebagai input untuk fungsi F. Fungsi F juga menerima angka bulat sebagai input. Kata ketiga di-XOR dengan output pertama dari fungsi F, kemudian diputar satu bit ke arah kanan. Sementara itu, kata keempat diputar satu bit ke arah kiri dan di-XOR dengan output kedua dari fungsi F. Pada akhirnya, posisi kedua kata tersebut ditukar untuk menghasilkan hasil yang diinginkan.

$$(F_{r,0}\, F_{r,1}) = F(F_{r,0} F_{r,1}\, r)$$

$$R_{r+1,0} = ROR\,(R_{r,2} \oplus F_{r,0}\, 1)$$

$$R_{r+1,1} = ROL\,(R_{r,3,1}\, 1) \oplus F_{r,1}$$

$$R_{r+1,2} = R_{r,0}$$

$$R_{r+1,3} = R_{r,1} \quad \text{..... (2.6)}$$

Pada persamaan 2.6, untuk setiap iterasi putaran, di mana r = 0, ..., 15, digunakan fungsi ROR dan ROL untuk memutar kata pertama (32-bit) ke kanan atau kiri sesuai dengan jumlah bit yang ditentukan oleh argumen kedua. Tahap *whitening* pada keluaran membatalkan 'pertukaran' yang terjadi pada putaran terakhir, dan melakukan operasi XOR antara kata-kata keluaran dengan empat kata kunci yang telah diperluas [19].

### 2.2.5 Cara Kerja *Twofish*

Pada algoritma kriptografi *twofish* sendiri terdapat beberapa istilah yang perlu diketahui, seperti enkripsi, dekripsi, *plaintext*, *key*, *chipertext*, dan lain-lain. Algoritma *twofish* sendiri berperan untuk menyamarkan *plaintext* pada saat proses enkripsi dan mengembalikan *chipertext* ke *plaintext* pada saat proses dekripsi. Secara sederhana, pada saat proses enkripsi dilakukan, *plaintext* akan disamarkan dengan bantuan *key* yang akan di-*generate* oleh pengguna, setelah itu, algoritma *twofish* akan mengacak *plaintext* tersebut beserta dengan kuncinya sehingga dihasilkan *chipertext*. Untuk ilustrasi, seperti pada gambar 2.3.


**Gambar 2.3 Cara kerja sederhana enkripsi menggunakan *twofish***

Pada gambar 2.3 secara sederhana ditampilkan cara kerja algoritma *twofish* pada saat melakukan enkripsi, *plaintext* atau teks yang akan dienkripsi akan diacak oleh algoritma *twofish* Bersama dengan *key* sebanyak 16 kali putaran pengacakan, setelah dilakukan pengacakan, *plaintext* yang sebelumnya dapat dengan mudah untuk dibaca, akan berganti menjadi *chipertext* yang sulit untuk dibaca dan diketahui, hal ini bertujuan agar pihak luar selain pengirim dan penerima tidak dapat mengetahui pesan tersebut [25].

Pada algoritma *twofish* terdiri dari tiga fase utama, yaitu penjadwalan kunci, tahap enkripsi dan tahap dekripsi. Kemudian, di dalam penjadwalan kunci terdapat penjelasan sebagai berikut:

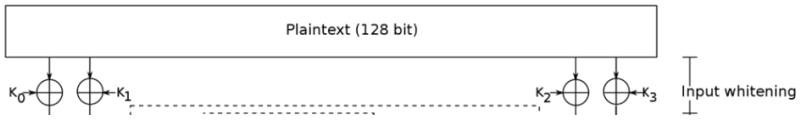
1. Tahapan paling awal sebelum melakukan enkripsi adalah perlu melalui tahap penjadwalan kunci. Pada algoritma *twofish* sendiri menggunakan kunci dengan panjang 128, 192, dan 256-bit. Jika panjang kunci dibawah kriteria tersebut, maka akan ditambahkan *zero byte* hingga ukuran kunci *twofish* terpenuhi.
2. Setelah tahap penjadwalan kunci selesai, kunci kemudian dibagi menjadi beberapa vektor, yakni vektor Me, Mo, dan vektor S. Vektor Me dan Mo berfungsi sebagai *list* dalam operasi fungsi h, sementara vektor S digunakan dalam proses enkripsi pada fungsi g.
3. Setelah itu, masing-masing kata kunci Kj yang diekspansi ke dalam fungsi h yaitu 2i dan 2i+1 akan melalui permutasi q0 dan q1 yang akan dilanjutkan dengan matriks MDS.
4. Hasil dari proses PHT akan keluar kata 2i, sedangkan kata yang dihasilkan melalui proses PHT akan dirotasi sejauh 8 bit ke kiri untuk menghasilkan kata 2i+1. Dari hasil 2i dan 2i+1 tersebut akan menjadi kunci yang sudah terjadwal.

Langkah-langkah yang terdapat di dalam proses enkripsi dan dekripsi *twofish* yaitu sebagai berikut:

1. Data masukan berupa *plaintext* dengan panjang 128 bit akan dipecah menjadi beberapa kelompok, yaitu P0, P1, P2, dan P3, di mana masing-masing kelompok memiliki panjang 32 bit. Pembagian ini juga dilakukan secara simetris, di mana P0 dan P1 ditempatkan di sisi kiri, sedangkan P2 dan P3 ditempatkan di sisi kanan.
2. *Plaintext* akan menjalani proses *whitening* pada tahap awal, di mana langkah ini melibatkan operasi XOR dengan empat kelompok kunci yang telah dijadwalkan sebelumnya, yaitu K0, K1, K2, dan K3. Untuk persamaan bisa dilihat pada persamaan 2.7

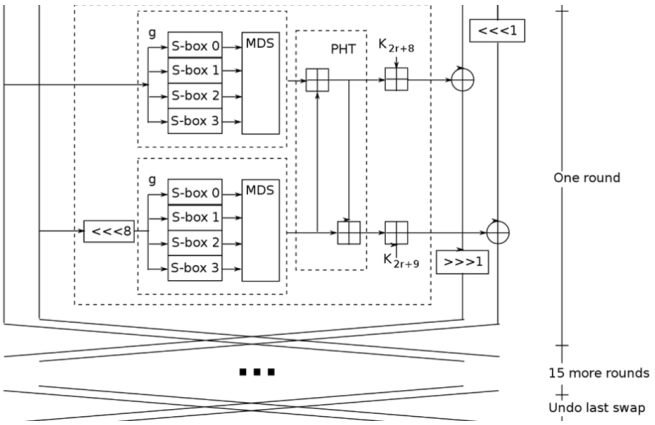
   $R_{0,i} = P_{i\ xor}\ K_i$ ……(2.7)

Untuk memudahkan pemahaman, dapat dilihat ilustrasi tahap *whitening* seperti terlihat pada gambar 2.4.



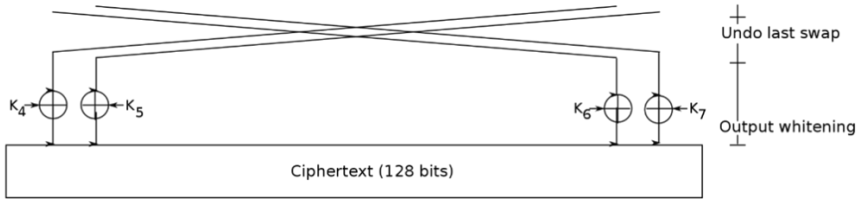

**Gambar 2.4 Proses *input whitening* [22]**

3. Tahap selanjutnya, masukan akan melalui tahap pada fungsi F yang meliputi fungsi G di dalamnya. Setelah itu akan melalui tahap transformasi Pseudo-Hadamard atau PHT, yang nantinya akan ditambahkan hasil PHT tersebut dengan kunci algoritma *twofish*. Tahap fungsi F akan dilakukan secara bertahap. Hasil dari fungsi F yang telah di-*whitening* akan menjadi R0 dan R1.
4. Dari R0 dan R1 tersebut akan dimasukan ke dalam fungsi G. Khusus untuk R1 tersebut sebelum dimasukan ke dalam fungsi G akan dipindahkan ke kiri sejauh 8-bit. Setelah itu, R0 dan R1 akan dikalikan dengan matriks MDS melalui *S-Boxes*. Hadil dari fungsi G tersebut akan menjadi T0 dan T1.
5. Kemudian dari T0 dan T1 tersebut akan diproses melalui PHT dari penggabungan T0 dan T1 dengan persamaan T0 + T1 dan T0 + 2T1. Kemudian hasil dari PHT tersebut akan ditambahkan dengan kunci yang telah dijadwalkan yaitu K2r+8 dan K2r+9. Hasil dari fungsi F tersebut dinamakan dengan F0 dan F1, ketika hasil tersebut sudah didapatkan, maka fungsi F sudah selesai. Untuk ilustrasi dapat dilihat pada gambar 2.5.




**Gambar 2.5 Ilustrasi Fungsi F [22]**

6. Setelah fungsi F telah terpenuhi, maka masing-masing F0 dan F1 akan di-XOR dengan R2 dan R3. Khusus untuk R2 tersebut akan di-XOR dengan F0 dan dirotasi ke kanan sejauh 1-bit. Untuk R3, kebalikan dari R2, yaitu akan dirotasi ke kiri sejauh 1-bit.
7. Kemudian ketika F0 dan F1 sudah di-XOR dengan R2 dan R3, maka akan diputar atau diiterasi sebanyak 16 kali, pada setiap putaran juga dilakukan seperti pada putaran sebelumnya.
8. Hasil dari perputaran tersebut akan menghasilkan enkripsi yang dinamakan *output whitening* dan blok terakhir akan di-*swap* dengan menukar bagian kanan dan kiri yang di-*undo.* Kemudian, hasil tersebut mendapatkan K4, K5, K6, dan K7 seperti terlihat pada gambar 2.6




**Gambar 2.6 Pertukaran blok terakhir atau *output whitening* [22]**

Setelah *chipertext* dihasilkan, itu menandakan bahwa proses enkripsi telah selesai. *Chipertext* sulit untuk diartikan ke dalam bahasa manusia, oleh karena itu, perlu dilakukan proses dekripsi agar *chipertext* dapat dikembalikan ke dalam bentuk *plaintext* yang dapat dibaca dengan mudah. Proses dekripsi pada twofish hampir mirip dengan proses enkripsi sebelumnya, hanya saja urutannya berkebalikan. Langkah-langkah pertamanya termasuk *output whitening*, blok *swap*, 16 putaran dekripsi, dan *input whitening*. Pada tahap dekripsi, masukan melibatkan *chipertext* dan kunci. Untuk memulai proses dekripsi, kunci yang sama yang digunakan pada tahap enkripsi harus dimasukkan agar hasil *plaintext* sesuai dengan yang diharapkan, karena algoritma *twofish* menggunakan kunci simetris [26]. Untuk ilustrasi sederhana, berikut proses sederhana dekripsi algoritma *twofish* seperti terlihat pada gambar 2.7.


**Gambar 2.7 Cara kerja sederhana dekripsi menggunakan *twofish***

Perbedaan blok diagram enkripsi dan dekripsi *twofish* adalah pada proses *input whitening*. Pada enkripsi *twofish*, proses *input whitening* menggunakan kunci yang di-XOR dengan data. Pada dekripsi *twofish*, proses *input whitening* menggunakan kunci yang di-XOR dengan data terenkripsi. Perbedaan lainnya adalah pada arah aliran data. Pada enkripsi *twofish*, data mengalir dari kiri ke kanan. Pada dekripsi *twofish,* data mengalir dari kanan ke kiri.

### 2.2.6 Tabel ASCII

Tabel ASCII (*American Standard Code for Information Interchange*) adalah suatu daftar yang mencakup seluruh huruf dalam alfabet Romawi, serta beberapa karakter tambahan. Setiap karakter dalam tabel tersebut selalu memiliki representasi kode yang konsisten. Sebagai contoh, huruf 'b kecil' selalu diwakili oleh kode 98. Jika diubah menjadi bentuk biner, kode untuk 'b kecil' menjadi 110 0010 dengan mewakili 9 dan 1 [27]. Berikut adalah contoh tabel ASCII seperti terlihat pada gambar 2.8.

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | File separator | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | Group separator | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | Record separator | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |

**Gambar 2.8 Tabel ASCII [22]**