KOMBINASI STEGANOGRAFI BIT *MATCHING* DAN KRIPTOGRAFI DES UNTUK PENGAMANAN DATA

Tesis

untuk memenuhi sebagian persyaratan

mencapai derajat Sarjana S-2 Program Studi

Magister Sistem Informasi

Budi Prasetiyo

24010411400009

PROGRAM PASCASARJANA
UNIVERSITAS DIPONEGORO
SEMARANG
2013

KOMBINASI STEGANOGRAFI BERBASIS BIT *MATCHING* DAN KRIPTOGRAFI DES UNTUK PENGAMANAN DATA

ABTRAK

Pada penelitian ini dilakukan kombinasi steganografi dan kriptografi untuk pengamanan data dengan tidak mengubah kualitas media *cover*. Metode steganografi yang digunakan dengan melakukan pencocokan bit pesan pada bit MSB citra. Proses pencocokan dilakukan secara *divide and conquer*. Hasil indeks posisi bit kemudian dienkripsi menggunakan algoritma kriptografi *Data Encryption Standard* (DES).

Masukkan data berupa pesan teks, citra, dan kunci. *Output* yang dihasilkan berupa chiperteks posisi bit yang dapat digunakan untuk merahasiakan data. Untuk mengetahui isi pesan semula diperlukan kunci dan citra yang sama.

Kombinasi yang dihasilkan dapat digunakan untuk pengamanan data. Kelebihan metode tersebut citra tidak mengalami perubahan kualitas dan kapasitas pesan yang disimpan dapat lebih besar dari citra. Hasil pengujian menunjukkan citra hitam putih maupun *color* dapat digunakan sebagai *cover*, kecuali citra 100% hitam dan 100% putih. Proses pencocokan pada warna citra yang bervariasi lebih cepat. Kerusakan pesan dengan penambahan *noise salt and peper* mulai terjadi pada nilai MSE 0,0067 dan *gaussian* mulai terjadi pada nilai MSE 0,00234.

Kata kunci: steganografi, pencocokan bit, *divide and conquer*, indeks bit, MSB, enkripsi, dekripsi, DES.

COMBINING STEGANOGRAPHY BIT MATCHING AND CRYPTOGRAPHY DES FOR DATA SECURITY

ABSTRACT

In this research, has been discussed about combination of steganography and cryptography to secure data without change the quality of cover medium. Steganographic methods used to matching the message bits on the MSB bit of image cover. Matching process is done by divide and conquer method. It will be result a bit position index, then encrypted using cryptographic DES (Data Encryption Standard).

The input are text message, image, and key. The output is ciphertext bit index which can be used to secure the messages. To read the contents of the message, we require the same image cover and key.

Outcomes of proposed method can be used to secure the data. The advantages of this method, such as the image quality has not changed and the capacity of stored messages can be larger than the image. Acoording to the research that grayscale or color images can be used as a image cover, except the image contain 100% black and 100% white. Bit matching process on image which have much variety of color need less time. The damage of messages in addition of noise salt and pepper start from 0,0067 of MSE value and gaussian strat from MSE 0,00234.

Keywords: steganography, bits matching, divide and conquer, bit index, MSB, encryption, decryption, DES.

BAB I

BAB I PENDAHULUAN

1.1. Latar Belakang

Seiring perkembangan zaman, kebutuhan manusia akan informasi semakin meningkat. Ditengah-tengah perkembangan teknologi informasi yang kian semarak, internet tidak lagi menjamin penyediaan informasi yang aman. Berbagai mesin-pencari (*search-engine*) terus berkembang ditambah dengan serangan *virus*, penyadap, *spam* maupun *hacker* yang menjamur dapat mencuri data-data bersifat rahasia (Kautzar, 2007). Mengatasi hal tersebut berbagai cara untuk meningkatkan keamanan data terus dikembangkan, diantaranya kriptografi dan steganografi.

Steganografi adalah seni dan ilmu menyembunyikan data pada media lain sebagai *cover* (misalnya citra) sehingga terlihat samar (Provos dan Honeyman, 2003). Kriptografi adalah seni dan ilmu menjaga kerahasiaan data (Scneicher, 1996). Pada kriptografi, data asli diubah menjadi bentuk lain yang tidak dapat dibaca. Penggabungan steganografi dan kriptografi secara bersamaan dapat meningkatkan pengamanan data (Krenn, 2004).

Metode penggabungan steganografi dan kriptografi banyak dikembangkan. Pada umumnya teknik yang digunakan yaitu dengan mengenkripsi pesan terlebih dahulu (kriptografi), kemudian menyisipkannya ke media *cover* (steganografi) (Raphael dan Sundaram, 2011). Namun, proses penyisipan dapat berpengaruh pada kualitas media *cover* tersebut.

Upaya untuk meminimalisir perubahan kualitas *cover* dapat dilakukan dengan penyisipan pada bit terakhir (*least significant bit*). Perubahan kualitas *cover* tidak tampak kasat mata (Chan dan Cheng, 2003), tetapi penyisipan pada bit terakhir dapat mengakibatkan pesan rusak ketia citra dikompresi (Nikolaidis dan Pitas, 1998). Ketahanan terhadap *robust* dapat dilakukan dengan pemilihan pada bit pertama (*most significant bit*), tetapi justru mengakibatkan perubahan kualitas *cover* menjadi tampak dan dapat dicurigai.

Challita dan Farhat (2011) mengembangkan cara lain penggabungan steganografi dan kriptografi tanpa mengubah medi 1 Teknik yang dilakukan yaitu den cocokan bit pesan pada *cover*, kemudian di proses enkripsi (kriptografi). Sal goritma

kriptografi yang terkenal sejak 1977 dan menjadi standar adalah *Data Encryption Standard* (DES).

Pada penelitian ini akan dilakukan kombinasi steganografi dan kriptografi tanpa mengubah media *cover*. Metode steganografi yang digunakan berbasis pencocokan bit (*bit matching*) pada bit pertama (*most significant bit*) dan metode kriptografi yang digunakan yaitu algoritma DES.

1.2. Tujuan Penelitian

Tujuan penelitian ini adalah:

a. Memodifikasi penggabungan steganografi dan kriptografi tanpa mengubah kualitas citra.
b. Menghasilkan aplikasi pengamanan data dengan kombinasi steganografi berbasis bit *matching* dan kriptografi DES.

1.3. Manfaat Penelitian

Manfaat dari penelitian ini yaitu:

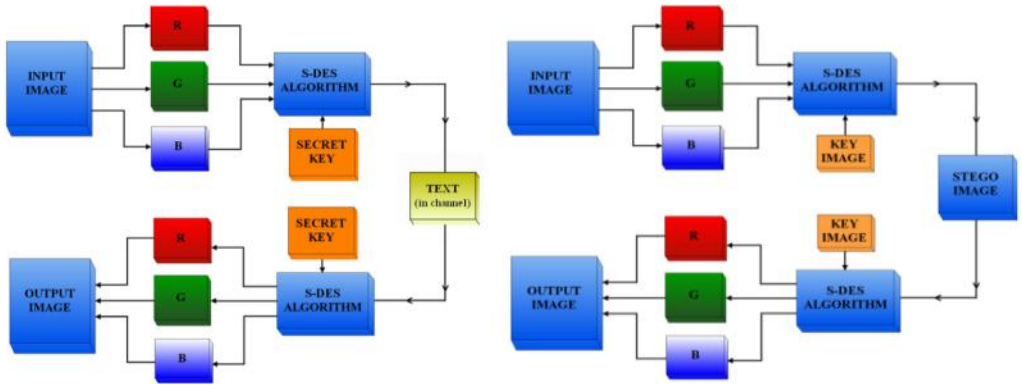
a. Menghasilkan model pengamanan data dengan penggabungan steganografi dan kriptografi DES.
b. Penelitian ini diharapkan dapat memberikan informasi ilmiah kepada mahasiswa dan masyarakat untuk menjaga kerahasiaan data.
c. Penelitian ini dapat dijadikan sebagai bahan referensi dan penelitian lanjut.

BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1. Tinjauan Pustaka

Terdapat beberapa literatur jurnal internasional terkait dengan perkembangan metode steganografi dan kriptografi terhadap kasus yang ada. Banyak metode yang telah dikembangkan untuk pengamanan data. Pada umumnya teknik yang digunakan yaitu dengan mengenkripsi pesan terlebih dahulu (proses kriptografi), kemudian menyisipkannya ke media *cover* (proses steganografi) (Raphael dan Sundaram, 2011).

Teknik kombinasi tidak hanya terbatas seperti teknik Raphael. Narayana dan Prasad (2010) pada penelitianya mengkaji 2 pendekatan untuk mengamankan media *cover* steganografi dengan melakukan enkripsi. Metode pertama, citra steganografi dienkripsi langsung dengan S-DES, hasilnya berupa chiperteks. Metode kedua, dengan mengenkripsi citra kemudian hasil chiperteks disisipkan pada citra lain.



Gambar 0.1 Metode kombinasi Narayana dan Prasad
(a) Metode pertama; (b) Metode kedua.

Salah satu metode steganografi yang paling mudah adalah LSB (Least *Significant Bit*). Langkah digunakan yaitu dengan menyisipkan bit terakhir (*least*) pada setiap *pixel* dengan bit pesan. Terminologi LSB dikaji oleh Sharp, 2001. Penyisipan pada LSB akan merubah nilai bit, tetapi tidak tampak kasat mata, sehingga pihak ketiga tidak mengetahui adanya pesan rahasia dibalik media *cover* (Chan dan Cheng, 2003).

Penggunaan LSB pada ko[3]teganografi dan kriptografi dilak[3]ı, 2010. Proses tersebut terdiri dari 3 tah[3]enkripsi, steganografi dan, dekri[3]ipsi dan dekripsi dilakukan dengan algoritma DES (*Data Encryption Standard)*. Penggunaan LSB

dapat meminimalisir perubahan kualitas citra, namun kapasitas pesan yang dapat ditampung terbatas sesuai ukuran citra. Kekre, 2008 melakukan penelitian steganografi LSB untuk meningkatkan kapasitas pesan dengan pendekatan PVD (*Pixel value Differencing*). Penyisipan LSB berdasarkan perbandingan besaran nilai bit MSB (*Most Significant Bit*). Jika nilai 4 bit MSB pertama "1", maka disisipkan sekaligus pada 4 bit terakhir. Jika 3 bit MSB pertama "1", maka disisipkan pada 3 bit terakhir. Jika 2 bit MSB pertama "1", maka disisipkan pada 2 bit terakhir. Sedangkan jika diluar kriteria, maka penyisipan dilakukan pada bit terakhir (*least*).

Kualitas citra merupakan komponen penting dalam steganografi. Challita dan Farhat (2011) mengembangkan cara lain penggabungan steganografi dan kriptografi tanpa mengubah kualitas citra. Teknik yang dilakukan yaitu dengan mencocokan bit pesan pada *cover*, hasil pencocokan berupa indeks posisi bit. Indeks inilah yang kemudian dienkripsi. *Outputnya* berupa chiperteks indeks bit. Pencocokan bit dilakukan secara *divide and conquer* yang teridiri dari 3 proses, yaitu *divide*, *conquer*, dan *combine* (Cormen dkk., 2009). Susunan bit yang panjang dipecah menjadi dua bagian kecil (*divide*), kemudian mencocokan setiap bagian tersebut (*conquer*). Hasil solusi tiap bagian kemudian digabung menjadi solusi total (*combine*).

Secara umum hasil penelitian yang sudah dilakukan sebelumnya terkait kasus pada penelitian ini ditunjukkan pada Tabel 2.1.

Tabel 0.1 Hasil penelitian yang berkaitan dengan steganografi dan kriptografi

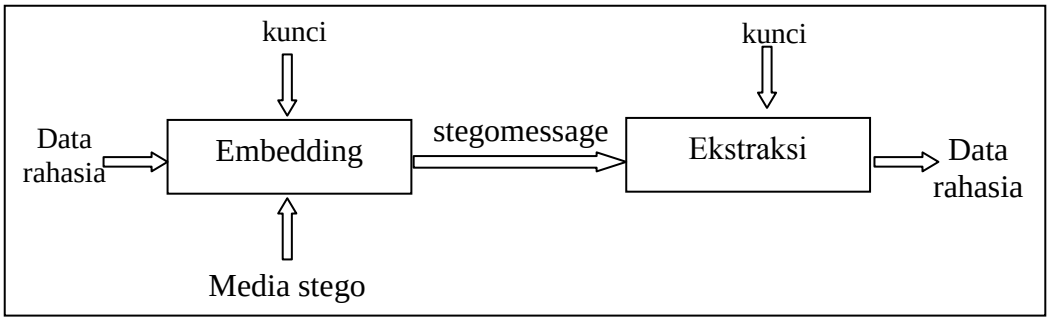
No	Peneliti	Hasil penelitian
1.	Raphael dan Sundaram, 2011	Penelitian tentang steganografi dan kriptografi.
2.	Narayana dan Prasad, 2010	Pengamanan citra steganografi dengan kriptografi S-DES.
3.	Sharp, 2001	Terminologi LSB.
4.	Chan dan Cheng, 2003	Steganografi dengan metode LSB.
5.	Seth, dkk., 2010	Penggabungan steganografi LSB dan kriptografi.
6.	Krenn, 2004	Penggabungan steganografi dan kriptografi dapat meningkatkan kemanan data.
7.	Kekre dkk., 2008	Penyisipan steganografi dengan PVD (*Pixel value Differencing*),yaitu penyisipan LSB bergantung pada besaran nilai MSB. Hal ini bertujuan untuk meningkatkan kapasitas pesan.
8.	Challita dan Farhat, 2011	Penggabungan steganografi dengan pencocokan bit dan kriptografi.
9.	Cormen dkk., 2009	Penyelesaian masalah secar *divide and conquer*.

2.2. Dasar Teori

2.2.1. Steganografi

Steganografi berasal dari bahasa Yunani yaitu, *Steganós* yang berarti menyembunyikan dan *Graptos* yang artinya tulisan, sehingga steganografi diartikan sebagai "tulisan tersembunyi (*covered writing*)". Steganografi adalah ilmu dan seni menyembunyikan pesan rahasia (*hiding message*) sedemikian sehingga keberadaan (eksistensi) pesan tidak terdeteksi oleh indera manusia (Munir, 2004).

Proses penyembunyian data kedalam media disebut penyisipan (*embedding*), sedangkan proses sebaliknya disebut ekstraksi. Secara umum proses tersebut ditunjukkan pada Gambar 2.2.



Gambar 0.2 Proses penyisipan dan ekstraksi dalam steganografi (Suharto, 2004)

2.2.1.1. Sejarah Steganografi

Steganografi sudah dikenal sejak 440 SM. Herodotus (sejarawan Yunani) menyebutkan dua contoh steganografi di dalam kisah "*Histories of Herodotus"*. Kisah pertama, pada saat itu, penguasa Yunani kuno, Histiaeus sedang ditawan oleh Raja Darius di Susa. Histiaeus ingin mengirim pesan rahasia kepada menantunya, Aristagoras, di Miletus. Oleh karena itu, Histiaeus mencukur habis rambut budaknya dan menuliskan (*tato*) pesan rahasia yang ingin dikirim di kepala budak tersebut. Setelah rambut budak tumbuh cukup lebat, barulah budak tersebut dikirim ke Miletus untuk menyampaikan pesan. Kisah kedua, Demeratus mengirimkan peringatan akan serangan Yunani yang selanjutnya dengan menuliskan pesan tersebut di atas sebuah papan kayu dan melapisinya dengan lilin. Lembaran pesan akan ditutup dengan lilin, untuk melihat isi pesan, pihak penerima harus memanaskan lilin terlebih dahulu (Johnson dan Jojadia, 2002).

Teknik steganografi sudah ada sejak 4000 tahun yang lalu di kota Menet Khufu, Mesir. Awalnya berupa penggunaan *hieroglyphic* yakni menulis menggunakan karakter-karakter dalam bentuk gambar. Ahli tulis menggunakan tulisan Mesir kuno ini untuk menceritakan kehidupan majikannya. Tulisan Mesir kuno tersebut menjadi ide untuk membuat pesan rahasia saat ini. Oleh karena itulah, tulisan Mesir kuno yang menggunakan gambar dianggap sebagai steganografi pertama di dunia (Ariyus, 2009).

Teknik steganografi yang lain adalah tinta yang tidak tampak (*invisible ink*) yaitu dengan menggunakan air sari buah jeruk, urin atau susu sebagai tinta untuk menulis pesan. Cara membacanya adalah dengan dipanaskan di atas api. Tinta yang sebelumnya tidak terlihat, ketika tekena panas akan menjadi gelap sehingga dapa dibaca. Teknik ini digunakan oleh bangsa Romawi yang juga digunakan pada Perang Dunia II oleh tentara Jerman (Kahn, 1996).

Bangsa Cina menggunakan cara yang berbeda pula, yaitu manusia sebagai media pembawa pesan. Orang itu akan dicukur rambutnya sampai botak dan pesan akan dituliskan di kepalanya. Kemudian pesan akan dikirimkan ketika rambutnya sudah tumbuh (Fabien, 1999).

2.2.1.2. Teknik Steganografi

Menurut Ariyus (2009), ada tujuh teknik dasar yang digunakan dalam steganografi, yaitu :

a. *Injection*, merupakan suatu teknik menanamkan pesan rahasia secara langsung ke suatu media. Salah satu masalah dari teknik ini adalah ukuran media yang diinjeksi menjadi lebih besar dari ukuran normalnya sehingga mudah dideteksi. Teknik ini sering juga disebut *embedding*.
b. Substitusi, data normal digantikan dengan data rahasia. Biasanya, hasil teknik ini tidak terlalu mengubah ukuran data asli, tetapi tergantung pada file media dan data yang akan disembunyikan. Teknik substitusi bisa menurunkan kualitas media yang ditumpangi.
c. *Trans*form *Domain*, teknik ini sangat efektif. Pada dasarnya, transformasi domain menyembunyikan data pada transform space.
d. *Spread Spectrum*, sebuah teknik pengtransmisian menggunakan *pseudonoise code*, yang independen terhadap data informasi sebagai modulator bentuk gelombang untuk menyebarkan energi sinyal dalam sebuah jalur komunikasi (*bandwidth*) yang lebih besar daripada sinyal jalur komunikasi informasi. Oleh penerima, sinyal dikumpulkan kembali menggunakan replika *pseudo-noise code* tersinkronisasi.
e. *Statistical Method*, teknik ini disebut juga skema *steganographic* 1 bit. Skema tersebut menanamkan satu bit informasi pada media tumpangan dan mengubah statistik walaupun hanya 1 bit. Perubahan statistik ditunjukkan dengan indikasi 1 dan jika tidak ada perubahan, terlihat indikasi 0. Sistem ini bekerja berdasarkan kemampuan penerima dalam membedakan antara informasi yang dimodifikasi dan yang belum.
f. *Distortion*, metode ini menciptakan perubahan atas benda yang ditumpangi oleh data rahasia.
g. *Cover Generation*, metode ini lebih unik daripada metode lainnya karena *cover* object dipilih untuk menyembunyikan pesan. Contoh dari metode ini adalah *Spam Mimic* (Ariyus, 2009).

2.2.1.3. Tujuan Steganografi

Tujuan dari steganografi adalah menyembunyikan keberadaan pesan dan dapat dianggap sebagai pelengkap dari kriptografi yang bertujuan untuk menyembunyikan isi pesan. Steganografi sebagai suatu teknik penyembunyian informasi pada data digital lainnya dapat dimanfaatkan untuk berbagai tujuan seperti :

a. *Tamper-proofing* dimana steganografi digunakan sebagai alat untuk mengidentifikasikan atau alat indikator yang menunjukkan data *host* telah mengalami perubahan dari aslinya.

b. *Feature location* dimana steganografi digunakan sebagai alat untuk mengidentifikasikan isi dari data digital pada lokasi-lokasi tertentu, seperti contohnya penamaan objek tertentu dari beberapa objek yang lain pada suatu citra digital.
c. *Annotation/caption* dimana steganografi hanya digunakan sebagai keterangan tentang data digital itu sendiri.
d. *Copyright-Labeling* dimana steganografi dapat digunakan sebagai metoda untuk penyembunyian label hak cipta pada data digital sebagai bukti otentik kepemilikan karya digital tersebut.

2.2.1.4. Media Steganografi

Steganografi menggunakan sebuah berkas yang disebut dengan *cover*, tujuannya sebagai kamuflase dari pesan yang sebenarnya. Steganografi membutuhkan dua properti: wadah penampung (*cover*) dan yang kedua adalah data atau pesan rahasiayang disembunyikan (*hiddentext*). Berkas hasil dari proses steganografi sering disebut sebagai berkas stego (*stegofile*) atau stego objek. Steganografi digital menggunakan media digital sebagai wadah penampung (Christian, 1998).

2.2.2. Kriptografi

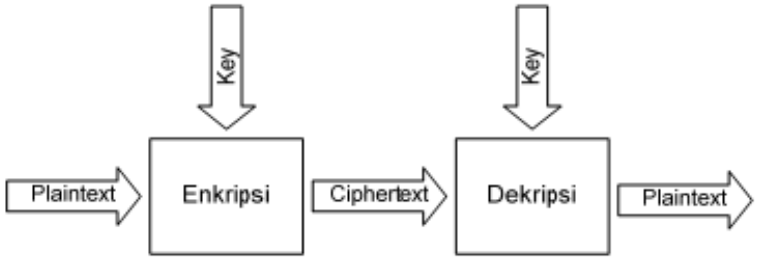
Kriptografi mempunyai peranan penting dalam dunia komputer. Hal ini disebabkan karena banyaknya informasi bersifat rahasia yang disimpan dan dikirmkan melalui media-media komputer. Informasi- informasi ini biasanya berisikan dokumen-dokumen penting dan data keuangan dari suatu instansi yang tidak ingin dibaca oleh pihak lain yang tidak berhak atas informasi tersebut. Oleh sebab itu ilmu kriptografi setiap saat terus dikembangkan oleh orang untuk dapat menjaga keamanan dan kerahasiaan informasi- informasi tersebut.

2.2.2.1. Definisi Kriptografi

Kriptografi berasal dari dua kata Yunani, yaitu *Crypto* yang berarti rahasia dan *Grapho* yang berarti menulis. Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi, seperti kerahasiaan data, keabsahan data, integritas data, serta autentikasi data (Menezes dkk., 1996). Secara umum kriptografi dapat diartikan sebagai ilmu dan seni penyandian yang bertujuan untuk menjaga keamanan dan kerahasiaan suatu pesan. Kriptografi pada dasarnya sudah dikenal sejak lama. Menurut catatan sejarah, kriptografi sudah digunakan oleh bangsa Mesir sejak 4000 tahun yang lalu oleh raja-raja Mesir pada saat perang untuk mengirimkan pesan rahasia kepada panglima perangnya melalui kurir-kurinya.

Orang yang melakukan penyandian ini disebut kriptografer, sedangkan orang yang mendalami ilmu dan seni dalam membuka atau memecahkan suatu algoritma kriptografi tanpa harus mengetahui kuncinya disebut kriptanalis. Kriptologi (*cryptology*) adalah ilmu yang mencakup kriptografi dan kriptoanalisis.

Cryptographic system atau *cryptosystem* adalah suatu fasilitas untuk mengkonversikan plainteks ke chiperteks dan sebaliknya. Kriptografi pada dasarnya terdiri dari dua proses, yaitu proses enkripsi dan proses dekripsi. Proses enkripsi adalah proses penyandian pesan terbuka menjadi pesan rahasia (chiperteks). Chiperteks inilah yang nantinya akan dikirimkan melalui saluran komunikasi terbuka. Pada saat chiperteks diterima oleh penerima pesan, maka pesan rahasia tersebut diubah lagi menjadi pesan terbuka melalui proses dekripsi sehingga pesan tadi dapat dibaca kembali oleh penerima pesan. Secara umum, proses enkripsi dan dekripsi dapat dilihat seperti pada Gambar 2.3.



Gambar 0.3 Proses enkripsi dan dekripsi (Munir, 2006)

Pesan terbuka (plainteks) diberi lambang M, yang merupakan singkatan dari *"Message"*. Plainteks ini dapat berupa teks, foto, atau video yang berbentuk data biner. Plainteks inilah yang nantinya akan dienkripsi menjadi pesan rahasia (chiperteks) yang dilambangkan dengan C. Secara matematis, operasi enkripsi dan dekripsi dapat diterangkan sebagai berikut:

$$EK(M) = C \text{ (Proses Enkripsi)} \quad (2.1)$$

$$DK(C) = M \text{ (Proses Dekripsi)} \quad (2.2)$$

Pada saat proses enkripsi kita menyandikan pesan M dengan suatu kunci K lalu dihasilkan pesan C. Sedangkan pada proses dekripsi, pesan C tersebut diuraikan dengan menggunakan kunci K sehingga dihasilkan pesan M yang sama seperti pesan sebelumnya.

2.2.2.2. Sejarah Kriptografi

Kriptografi mempunyai sejarah yang panjang dan menakjubkan. Informasi yang lengkap mengenai sejarah kriptografi dapat ditemukan di dalam buku David Kahn yang

berjudul *The Codebreakers*. Buku ini menulis secara rinci sejarah kriptografi, mulai dari penggunaan kriptografi oleh Bangsa Mesir 4000 tahun yang lalu (berupa *hieroglyph* pada piramid) hingga penggunaan kriptografi abad ke-20 (Ariyus, 2009).

Sebagian besar sejarah kriptografi merupakan kriptografi klasik, yaitu metode kriptografi yang menggunakan kertas dan pensil atau menggunakan alat bantu mekanik yang sederhana. Kriptografi klasik secara umum dikelompokkan menjadi dua kategori, yaitu algoritma transposisi (*transposition cipher)* dan algoritma substitusi *(substitution cipher*). Algoritma transposisi adalah algoritma yang mengubah susunan-susunan huruf di dalam pesan, sedangkan algoritma substitusi yaitu mengganti setiap huruf atau kelompok huruf dengan sebuah huruf atau kelompok huruf yang lain.

Penggunaan *transposition cipher* yaitu oleh tentara Sparta di Yunani pada permulaan tahun 500 SM. Mereka menggunakan apa yang dinamakan *scytale* (Gambar 2.4 (a)). *Scytale* terdiri dari sebuah kertas panjang dari daun *papyrus* yang dililitkan pada sebuah silinder dari diameter tertentu (diameter dari silinder merupakan kunci dari penyandian tersebut). Pesan ditulis baris per baris dan secara horisontal (Gambar 2.4. (b)). Apabila pita dilepas, maka setiap huruf akan tersusun secara acak membentuk pesan rahasia (pesan yang tidak dapat dibaca). Agar pesan tersebut dapat dibaca, maka pesan tersebut harus kembali dililitkan ke silinder yang diameternya sama dengan diameter silinder pengirim.

(a) (b)

Gambar 0.4 (a) Sebuah *scytale*; (b) Pesan ditulis secara baris per baris (Munir, 2006)

2.2.2.3. Tujuan Kriptografi

Ada empat tujuan mendasar dari kriptografi yang juga merupakan aspek keamanan informasi (Munir, 2006), yaitu:

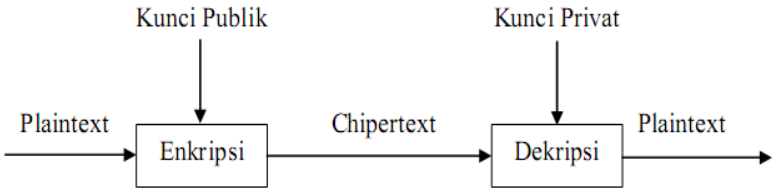
a. Kerahasiaan, adalah aspek yang berhubungan dengan penjagaan isi informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka informasi yang telah dienkripsi.
b. Integritas data, adalah aspek yang berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
c. Autentikasi, adalah aspek yang berhubungan dengan identifikasi atau pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.
d. Non-repudiation (menolak penyangkalan), adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman suatu informasi oleh yang mengirimkan, atau harus dapat membuktikan bahwa suatu pesan berasal dari seseorang, apabila ia menyangkal mengirim informasi tersebut.

2.2.2.4. Jenis Kriptografi

Algoritma kriptografi adalah suatu fungsi matematis yang digunakan untuk melakukan enkripsi dan dekripsi. Ada dua macam algoritma kriptografi, yaitu algoritma simetri (*symmetric algorithms*) dan algoritma asimetri (*asymmetric algorithms*).

2.2.2.4.1. Kriptografi Kunci Asimetri (Kriptografi Kunci Publik)

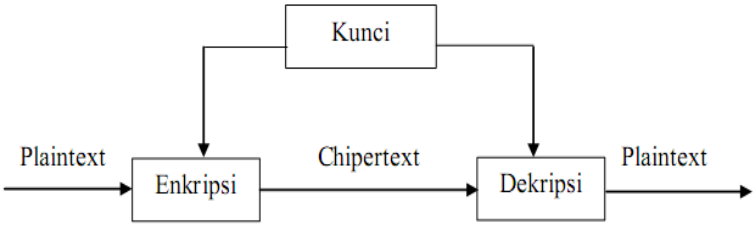
Kriptografi kunci asimetri yang sering disebut juga kriptografi kunci publik adalah algoritma yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsinya. Algoritma asimetri ini disebut kunci publik karena kunci untuk enkripsi dapat dibuat publik yang berarti semua orang boleh mengetahuinya. Sembarang orang dapat menggunakan kunci enkripsi tersebut untuk mengenkrip pesan namun hanya orang tertentu yaitu calon penerima pesan dan sekaligus pemilik kunci dekripsi yang merupakan pasangan kunci publik, yang dapat melakukan dekripsi terhadap pesan tersebut. Dalam sistem ini, kunci enkripsi disebut kunci publik, sementara kunci dekripsi sering disebut kunci privat. Contoh penggunaan kunci pada algoritma asimetri ditunjukkan pada Gambar 2.5.



Gambar 0.5 Enkripsi kunci asimetri

2.2.2.4.2. Kriptografi Kunci Simetri

Kriptografi simetri disebut juga sebagai kriptografi konvensional. Kriptografi simetri adalah algoritma kriptografi yang menggunakan kunci enkripsi yang sama dengan kunci dekripsinya. Kriptografi simetri sering disebut sebagai algoritma kunci rahasia, algortima kunci tunggal, atau algoritma satu kunci dan mengharuskan pengirim dan penerima menyetujui suatu kunci sebelum mereka dapat berkomunikasi dengan aman. Gambar 2.6 mengilustrasikan kinerja dari proses enkripsi kunci simetri.



Gambar 0.6 Kriptografi kunci simetri (Munir, 2006)

Kelebihan Kriptografi Simetri adalah sebagai berikut.

1. Proses enkripsi atau detesis kriptografi simetri membutuhkan waktu yang singkat.
2. Ukuran kunci simetri relatif lebih pendek.
3. Otentikasi pengiriman pesan langsung diketahui dari cipherteks yang diterima, karena kunci hanya diketahui oleh penerima dan pengirim saja.

Kekurangan Kriptografi Simetri adalah sebagai berikut.

1. Kunci simetri harus dikirim melalui saluran komunikasi yang aman, dan kedua entitas yang berkomunikasi harus menjaga kerahasiaan kunci.
2. Kunci harus sering diubah, setiap kali melakasanakan komunikasi.

Masalah utama yang dihadapi kriptografi simetri adalah membuat pengirim dan penerima menyetujui kunci rahasia tanpa ada orang lain yang mengetahuinya.

Salah satu cotoh algoritma kunci simetri adalah algoritma DES. DES sangat digunakan untuk melindungi data dalam dunia elektronika khususnya di bidang perbankan, finansial, dan e-*commerce*.

2.2.3. Algoritma *Data Encryption Standard* (DES)

Algoritma DES merupakan salah satu algoritma kriptografi simetri. Algoritma DES merupakan algoritma standar untuk kriptografi simetri. Pada sub bab ini penulis akan membahas dasar-dasar dan prinsip kerja dari algoritma DES itu sendiri.

Pertengahan tahun 1973, Pemerintah Amerika Serikat (AS) melalui *National Bureau of Standards* (NBS) mengumumkan kebutuhan akan suatu algoritma sandi yang akan digunakan sebagai standar untuk melindungi kerahasiaan dan keutuhan data-data penting baik yang sedang ditransmisikan maupun yang disimpan.

Algoritma DES merupakan salah satu proposal terbaik tahun 1977. Algoritma DES dikembangkan di IBM di bawah kepemimpinan W. L. Tuchman pada tahun 1972. Algoritma ini didasarkan pada algoritma *Lucifer* yang dibuat oleh *Horst Feistel*. Algoritma ini telah disetujui oleh *National Bureau of Standard* (NBS) setelah penilaian kekuatannya oleh *National Security Agency* (NSA) Amerika Serikat.

DES merupakan salah satu *chiper block* penyandian/kriptografi data yang populer dan telah dijadikan standard enkripsi kunci simetri sejak tahun 1976 dengan ukuran blok 64 bit dan ukuran kuncinya 56 bit. Algoritma DES dibuat di IBM, dan merupakan modifikasi dari algoritma terdahulu yang bernama *Lucifer*. *Lucifer* merupakan algoritma *cipher block* yang beroperasi pada blok masukan 64 bit dan kuncinya berukuran 128 bit (Munir, 2006).

2.2.3.1. Prinsip Kerja Algoritma DES

Sandi DES adalah hasil pengembangan dari Sandi *Feistel*, dengan demikian wajar bila terdapat Sandi *Feistel* didalamnya. Adapun prinsip kerja dari sandi DES adalah sebagai berikut (Munir, 2006).

1. Persiapan kunci, memecah kunci ke dalam 16 sub kunci.
2. Melakukan prosedur *Feistel* 16 iterasi, dengan menggunakan sub kunci yang telah disediakan, dan menggunakan fungsi yang telah ditentukan.

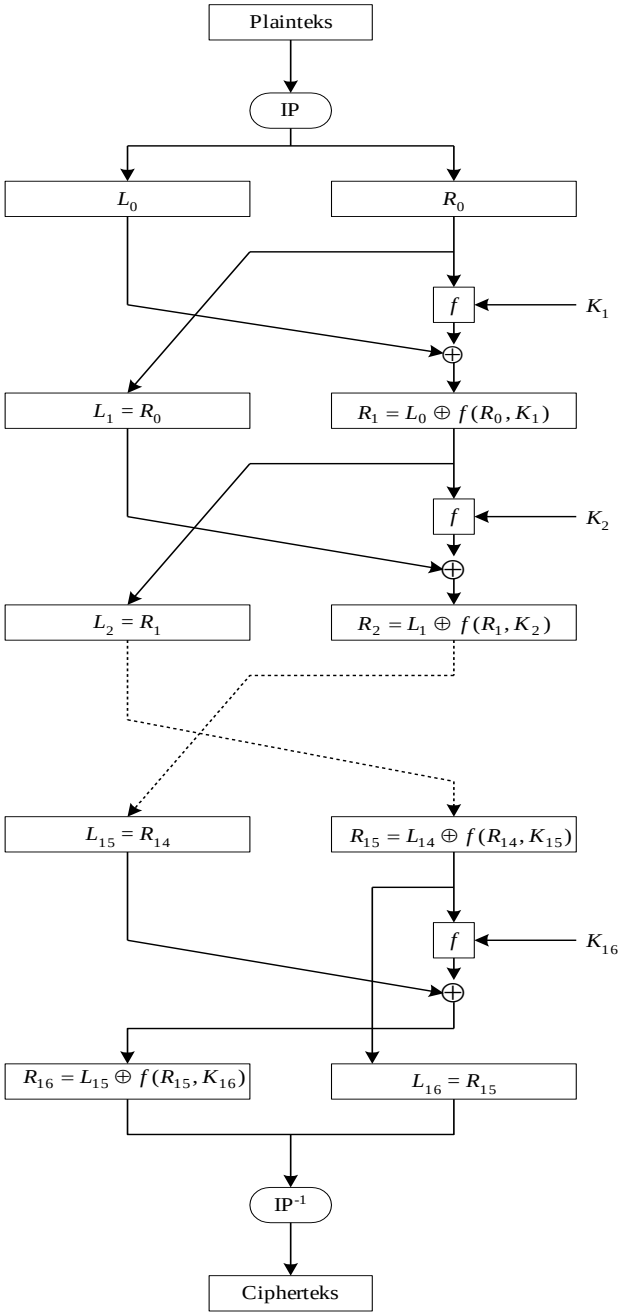
   Secara detail, yang dikerjakan oleh sandi *Feistel* sebagai berikut.

   a. Plainteks dikonversikan terlebih dahulu dalam biner, kemudian dibagi dan diproses per blok, dimana setiap blok terdiri dari 64 bit.
   b. Untuk setiap blok kemudian dilakukan koversi posisi terhadap Tabel IP (*Initial Permutation*), hasilnya kemudian dibagi menjadi 2 bagian, yaitu 32 bit pada bagian kiri disebut Lo, dan 32 bit di kanan disebut Ro.
   c. Kedua bagian ini kemudian dilakukan iterasi fungsi *f* sebanyak 16 kali (LiRi, $1<i<16$). Secara matematis, satu putaran DES dinyatatakan sebagai berikut: $Li = Ri - 1$, $Ri = Li - 1 \oplus f(Ri - 1, Ki)$ (2.3)

      Ki adalah kunci 48 bit yang terdiri dari 16 macam yang berbeda untuk setiap iterasi.
3. Hasil akhirnya kemudian dibalik, dan dioperasikan dengan invers dari IP (IP-1(R16,L16).

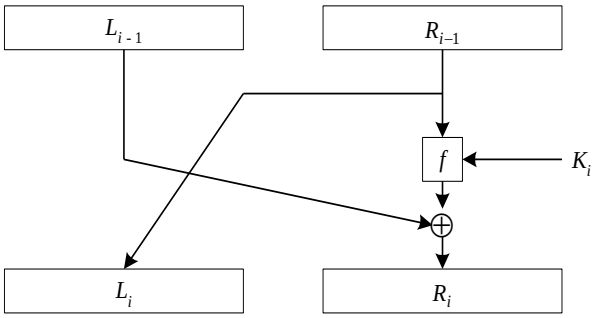
2.2.3.2. Skema Algoritma DES

Skema algoritma DES ditunjukkan pada Gambar 2.7.



Gambar 0.7 Algoritma enkripsi dengan DES (Schneier, 1996)

Satu putaran DES merupakan model jaringan *Feistel* (Gambar 2.8).



Gambar 0.8 Jaringan *Feistel* untuk satu putaran DES (Schneier, 1996)

Gambar 2.8 mengilustrasikan bahwa jika (L_{16}, R_{16}) merupakan keluaran dari putaran ke-16, maka (R_{16}, L_{16}) merupakan pra-chiperteks (*pre-ciphertext*) dari enkripsi ini. Chiperteks yang sebenarnya diperoleh dengan melakukan permutasi awal balikan, IP^{-1} terhadap blok pra-chiperteks.

2.2.3.3. Permutasi Awal

Sebelum putaran pertama, terhadap blok plainteks dilakukan permutasi awal (*initial permutation* atau IP). Tujuan permutasi awal adalah mengacak plainteks sehingga urutan bit-bit di dalamnya berubah. Pengacakan dilakukan dengan menggunakan matriks permutasi awal pada Tabel 2.2.

Tabel 0.2 Matriks permutasi awal

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Cara membaca tabel/matriks di atas: dua *entry* ujung kiri atas (58 dan 50) berarti:

"pindahkan bit ke-58 ke posisi bit 1",

"pindahkan bit ke-50 ke posisi bit 2", dan seterusnya.

2.2.3.4. Pembangkitan Kunci Internal

Pembangkitan kunci internal terjadi selama 16 putaran. Karena terdapat 16 putaran, maka dibutuhkan kunci internal sebanyak 16 buah, yaitu K_1, K_2, …, K_{16}. Kunci-kunci internal tersebut dapat dibangkitkan sebelum proses enkripsi atau bersamaan dengan proses enkripsi.

Kunci internal dibangkitkan dari kunci eksternal yang diberikan oleh pengguna. Kunci eksternal panjangnya 64 bit atau 8 karakter. Misalkan kunci eksternal yang tersusun dari 64 bit adalah *K*. Kunci eksternal ini menjadi masukan untuk permutasi dengan menggunakan matriks permutasi kompresi PC-1 yang ditunjukkan pada Tabel 2.3.

Tabel 0.3 Matriks PC-1

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Pada permutasi ini, tiap bit kedelapan (*parity bit*) dari delapan *byte* kunci diabaikan. Hasil permutasinya adalah sepanjang 56 bit, sehingga dapat dikatakan panjang kunci DES adalah 56 bit. Selanjutnya, 56 bit ini dibagi menjadi 2 bagian, kiri (C) dan kanan (D), yang masing-masing panjangnya 28 bit disimpan di dalam C_0 dan D_0:

C_0: berisi bit-bit dari *K* pada posisi:

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36

D_0: berisi bit-bit dari *K* pada posisi:

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

Selanjutnya, kedua bagian digeser ke kiri (*left shift*) sepanjang satu atau dua bit bergantung pada tiap putaran. Operasi pergeseran bersifat *wrapping* atau *round-shift,* aturan pergeseran setiap putaran ditunjukkan pada Tabel 2.4.

Tabel 0.4 Aturan pergeseran setiap putaran

Putaran, *i*	**Jumlah pergeseran bit**
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Misalkan (C_i, D_i) menyatakan penggabungan C_i dan D_i. (C_{i+1}, D_{i+1}) diperoleh dengan menggeser C_i dan D_i satu atau dua bit. Setelah pergeseran bit, (C_i, D_i) mengalami permutasi kompresi dengan menggunakan matriks PC-2 yang ditunjukkan pada Tabel 2.5.

Tabel 0.5 Matriks PC-2

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Dengan permutasi ini, kunci internal K_i diturunkan dari (C_i, D_i) yang dalam hal ini K_i merupakan penggabungan bit-bit C_i pada posisi:

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10

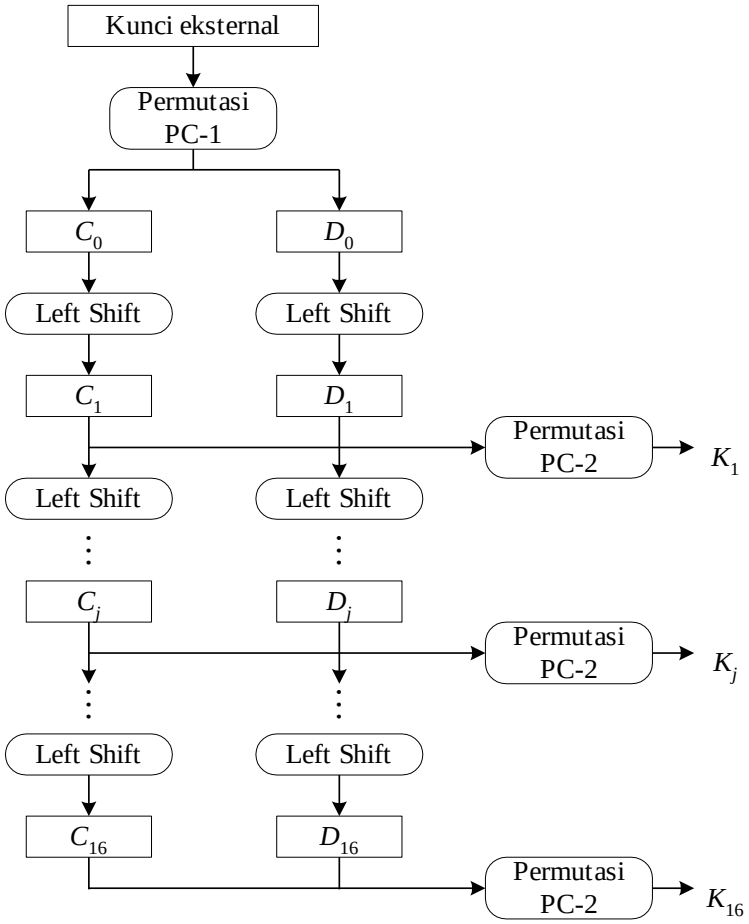
23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2

dengan bit-bit D_i pada posisi:

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48

44, 49, 39,56, 34, 53, 46, 42, 50, 36, 29, 32

Jadi, setiap kunci internal K_i mempunyai panjang 48 bit.



Gambar 0.9 Proses pembangkitan kunci-kunci internal DES (Schneier, 1996)

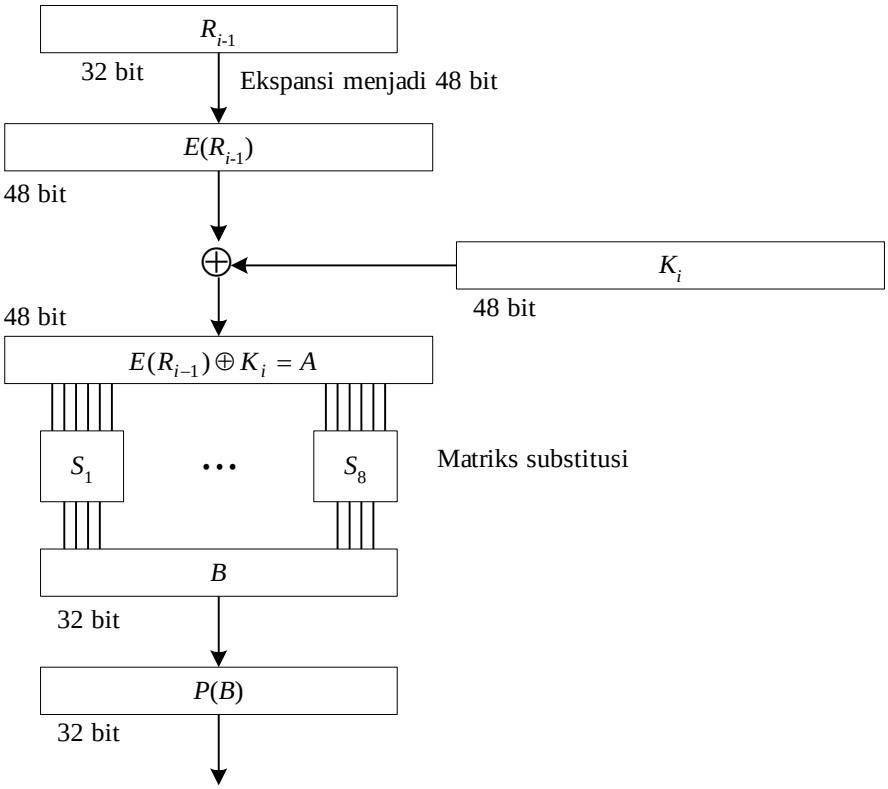
2.2.3.5. Enkripsi

Proses enkripsi terhadap blok plainteks dilakukan setelah permutasi awal Setiap blok plainteks mengalami 16 kali putaran enkripsi (Gambar 2.7). Setiap putaran enkripsi merupakan jaringan *Feistel* yang secara matematis dinyatakan sebagai berikut:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \qquad (2.4)$$

Diagram komputasi fungsi *f* diperlihatkan pada Gambar 2.10.



Gambar 0.10 Diagram komputasi fungsi *f* (Schneier, 1996)

Notasi *E* adalah fungsi ekspansi yang memperluas blok R_{i-1} yang panjangnya 32-bit menjadi blok 48 bit. Fungsi ekspansi direalisasikan dengan matriks permutasi ekspansi yang ditunjukkan pada Tabel 2.6.

Tabel 0.6 Matriks Permutasi Ekspansi

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Selanjutnya, hasil ekpansi, yaitu $E(R_{i-1})$, yang panjangnya 48 bit di-XOR-kan dengan K_i yang panjangnya 48 bit menghasilkan vektor *A* yang panjangnya 48-bit:

$$E(R_{i-1}) \oplus K_i = A \tag{2.5}$$

Vektor *A* dikelompokkan menjadi 8 kelompok, masing-masing 6 bit, dan menjadi masukan untuk proses substitusi.

2.2.3.6. Proses Subtitusi Menggunakan Kotak S-*box*

Proses substitusi dilakukan dengan menggunakan delapan buah kotak-S (*S-box*) yaitu S_1 sampai S_8. Setiap kotak-S menerima masukan 6 bit dan menghasilkan keluaran 4 bit.

Kelompok 6-bit pertama menggunakan S_1, kelompok 6-bit kedua menggunakan S_2, dan seterusnya.

Kotak-*S* di dalam algoritma DES adalah 6 × 4 *S-box* yang berarti memetakan 6 bit masukan menjadi 4 bit keluaran. Setiap S-*box* terdiri dari suatu tabel ukuran 4 × 6 (4 baris dan 16 kolom). Setiap baris diberi nomor 0 sampai 3 dan setiap kolom diberi nomor 0 sampai 15. Masukan untuk proses substitusi adalah 6 bit ($b_1b_2b_3b_4b_5b_{6)}$. Nomor baris dari tabel ditunjukkan oleh *string* bit b_1b_6 (menyatakan nilai 0 sampai 3 desimal). Nomor kolom ditunjukkan oleh *string* bit $b_2b_3b_4b_5$ (menyatakan nilai 0 sampai 15 desimal).

Contoh:

Misalkan masukan adalah 110100 dan salah satu kotak-*S* yang ada di dalam algoritma DES ditunjukkan pada Tabel 2.7 sebagai berikut

Tabel 0.7 Contoh proses subtitusi kotak pada kotak S-*box*

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

Nomor baris tabel = 10 (artinya baris 2 desimal). Nomor baris tersebut diperoleh dari yaitu b_1: 1 dan b_2: 0.

Nomor kolom tabel = 1010 (artinya kolom 10 desimal). Nomor kolom tersebut diperoleh dari b_2: 1, b_3: 0, b_4: 1 dan b_5: 0.

Jadi, substitusi untuk 110100 adalah *entry* pada baris 2 dan kolom 10, yaitu 4 desimal atau 0100 dalam bentuk biner.

Kedelapan kotak-S tersebut ditunjukkan pada Lampiran 8.

Keluaran proses substitusi adalah vektor *B* yang panjangnya 48 bit. Vektor *B* menjadi masukan untuk proses permutasi. Tujuan permutasi adalah untuk mengacak hasil proses substitusi kotak-S. Permutasi dilakukan dengan menggunakan matriks permutasi *P* (*P-box*) yang ditunjukkan pada Tabel 2.8.

Tabel 0.8 Matriks permutasi P

16	7	20	21	29	12	28	17	1	15	23	26	5	8	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

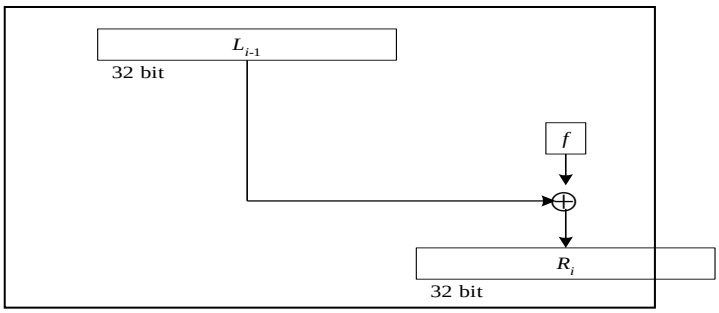
a. Bit-bit $P(B)$ merupakan keluaran dari fungsi f

b. Akhirnya, bit-bit $P(B)$ di-XOR-kan dengan L_{i-1} untuk mendapatkan R_i (lihat Gambar 2.11):

$$R_i = L_{i-1} \oplus P(B) \tag{2.6}$$

c. Jadi, keluaran dari putaran ke-*i* adalah

$$(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus P(B)) \tag{2.7}$$



Gambar 0.11 Skema perolehan R_i

2.2.3.7. Permutasi Akhir

Permutasi terakhir dilakukan setelah 16 kali putaran terhadap gabungan blok kiri dan blok kanan. Proses permutasi menggunakan matriks permutasi awal balikan (*inverse initial permutation* atau IP^{-1}). Matriks IP^{-1} ditunjukkan pada Tabel 2.10.

Tabel 0.9 Matriks IP^{-1}

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

2.2.3.8. Dekripsi

Proses dekripsi terhadap chiperteks merupakan kebalikan dari proses enkripsi. DES menggunakan algoritma yang sama untuk proses enkripsi dan dekripsi. Jika pada proses

enkripsi urutan kunci internal yang digunakan adalah K_1, K_2, …, K_{16}, maka pada proses dekripsi urutan kunci yang digunakan sebaliknya, yaitu K_{16}, K_{15}, …, K_1.

Masing-masing putaran 16, 15, …, 1, menghasilkan keluaran pada setiap putaran *deciphering* sebagai berikut:

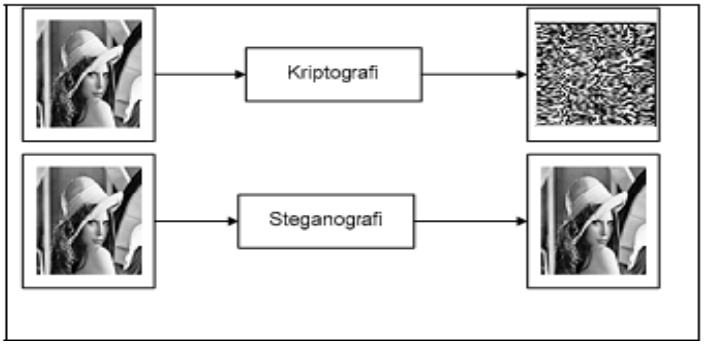
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2.8}$$

yang dalam hal ini, (R_{16}, L_{16}) adalah blok masukan awal untuk deciphering. Blok (R_{16}, L_{16}) diperoleh dengan mempermutasikan chiperteks dengan matriks permutasi IP^{-1}. Pra-keluaran dari *deciphering* adalah adalah (L_0, R_0). Dengan permutasi awal IP akan didapatkan kembali blok plainteks semula. Tinjau kembali proses pembangkitan kunci internal pada Gambar 2.9.

Selama *deciphering*, K_{16} dihasilkan dari (C_{16}, D_{16}) dengan permutasi PC-2. Tentu saja (C_{16}, D_{16}) tidak dapat diperoleh langsung pada permulaan *deciphering*. Tetapi karena (C_{16}, D_{16}) = (C_0, D_0), maka K_{16} dapat dihasilkan dari (C_0, D_0) tanpa perlu lagi melakukan pergeseran bit. Catatlah bahwa (C_0, D_0) yang merupakan bit-bit dari kunci eksternal K yang diberikan pengguna pada waktu dekripsi. Selanjutnya, K_{15} dihasilkan dari (C_{15}, D_{15}) yang diperoleh dengan cara menggeser C_{16} (yang sama dengan C_0) dan D_{16} (yang sama dengan C_0) satu bit ke kanan. Sisanya, K_{14} sampai K_1 dihasilkan dari (C_{14}, D_{14}) sampai (C_1, D_1). Catatlah bahwa (C_{i-1}, D_{i-1}) diperoleh dengan menggeser C_i dan D_i dengan cara yang sama yang ditunjukkan pada Tabel 2.4, tetapi pergeseran kiri (*left shift*) diganti menjadi pergeseran kanan (*right shift*).

2.2.4. Kelebihan dan Kekurangan Steganografi dan Kriptografi

Steganografi berbeda dengan kriptografi, di mana pihak ketiga dapat mendeteksi adanya data (*chipertext*), karena hasil dari kriptografi berupa data yang berbeda dari bentuk aslinya dan biasanya datanya seolah-olah berantakan, tetapi dapat dikembalikan ke bentuk semula. Ilustrasi mengenai perbedaan kriptografi dan steganografi ditunjukkan pada Gambar 2.12.



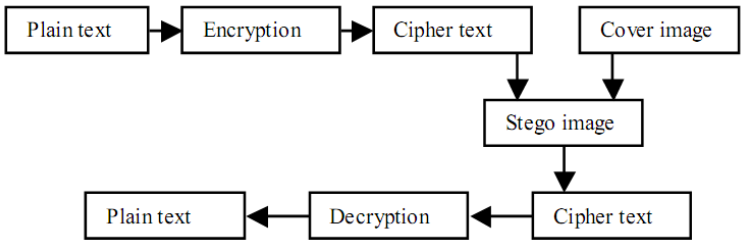
Gambar 0.12 Beda steganografi dan kriptografi (Munir, 2004)

Penyisipan pesan pada citra dapat mempengaruhi kualitas citra tersebut, walau tidak tampak secara kasat mata. Namun melalui pengolahan citra, dapat dideteksi pola-polanya sehingga dapat dicurigai dan memungkinkan untuk dilakukan penyerangan. Penggunaan kriptografi dengan mengubah menjadi bentuk lain memang sangat mudah dicurigai, tetapi pola perubahan pada kriptografi susah diketahui.

Kombinasi steganografi dan kriptografi dapat dilakukan untuk meningkatkan keamanan data (Krenn, 2004). Seandainya citra steganografi dapat dideteksi dan dibongkar, tetapi isinya bukan data asli melainkan data enkripsi dan untuk menhetahui isi pesan masih dipelukan kunci yang benar.

2.2.5. Kombinasi Steganografi dan Kriptografi

Kombinasi steganografi dan kriptografi pada umunya dilakukan dengan proses kriptografi terlebih dahulu kemudian steganografi, yaitu mengenkripsi pesan terlebih dahulu kemudian menyisipkan chiperteks hasil enkripsi ke media *cover* (Raphael dan Sundaram, 2011). Konsep penggabungan kriptografi dan steganografi ditunjukkan pada Gambar 2.13.



Gambar 0.13 Kombinasi Steganografi dan Kriptografi (Raphael dan Sundaram, 2011)

2.2.6. Pencocokan Bit secara *Divide and Conquer*

Pencocokan bit pesan pada bit citra dilakukan dengan metode *divide and conquer* yang terdiri dari 3 proses yaitu *divide*, *conquer* dan *combine*. Artinya memecah masalah menjadi beberapa bagian kecil (*divide*), kemudian secara rekursif menyelesaikan setiap masalah-masalah kecil tersebut (*conquer*). Selanjutnya solusi dari setiap masalah kecil tersebut hasilnya digabung menjadi satu solusi utama (*combine*) (Cormen dkk., 2009).

Penggunaan metode *divide and conquer* pada steganografi dilakukan oleh Challita dan Farhat (2011) dalam penelitiannya untuk mencocokan lokasi bit pesan dan bit citra pada proses penyisipan pesan. Misal diberikan sebuah barisan S1 dan S2, dan dinotasikan LCS(S1, S2) adalah algoritma untuk pencarian substring panjang (*longest common subsequence*) dari S1 yang muncul pada S2. Selanjutnya akan menghasilkan nilai *true* jika seluruh S1 terjadi pada S2. Ilustrasi algoritma (LCS) diberikan pada Gambar 2.14.

```
SPS(secretMessage, coverImage);
  if LCS(secretMessage, coverImage) is true,
       then
       store the positions of the indexes
       of the start and end bits of Secret
       that occur within Image the output
       file Output,
       else
       SPS(LeftPart-secretMessage,coverImage)
       SPS(RightPart-secretMessage,coverImage)
return Output,
```

Gambar 0.14 Algoritma pecocokan pesan pada citra (Challita dan Farhat, 2011)

BAB III
METODE PENELITIAN

3.1. Bahan dan Alat Penelitian

Bahan dan alat penelitian pada penelitian ini diuraikan sebagai berikut.

3.1.1. Bahan Penelitian

Bahan penelitian yang digunakan pada proses penelitian ini bersumber pada jurnal internasional, artikel ilmiah, dan buku-buku pendukung yang terkait dengan steganografi, dan kriptografi algoritma DES.

3.1.2. Alat Penelitian

Alat yang digunakan dalam penelitian ini terbagi 2 yaitu, perangkat keras (*hardware*) dan perangkat lunak (*software*).

1. Perangkat Keras

   Perangkat keras yang digunakan dalam penelitian ini adalah *notebook* dengan spesifikasi:

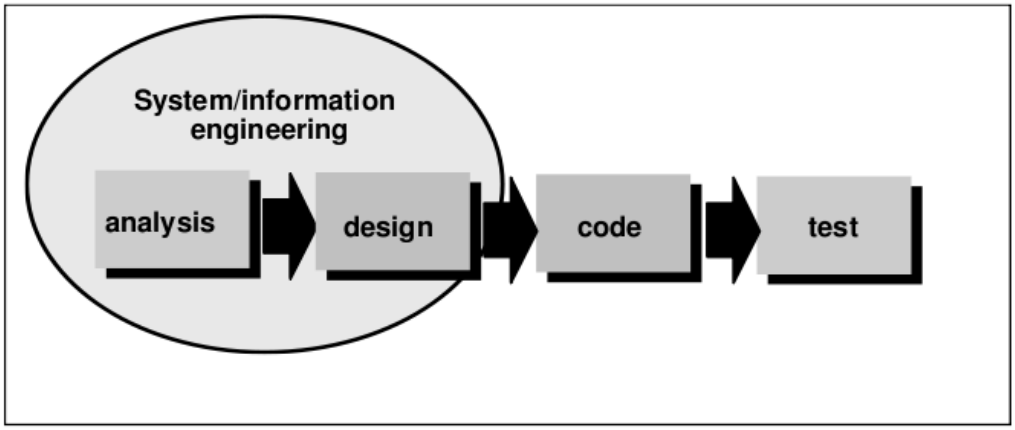
   a. *Processor* Intel® Core™ i3 2.40GHz
   b. Memori RAM 2 GB

2. Perangkat Lunak, yaitu

   a. Sistem operasi Windows
   b. *Software* MATLAB R2009a.

3.2. Jalan Penelitian

Metode yang digunakan dalam penelitian ini yaitu penggabungan steganografi dengan krtipografi. Algoritma kriptografi yang digunakan adalah DES. Terdapat 2 proses didalam steganografi, yaitu *embedding* dan ekstraksi. Pada penelitian ini akan dibangun suatu perangkat lunak stego-kripto dengan model *waterfall* yaitu suatu metode pengembangan *software* yang bersifat sekuensial. Metode *waterfall* ditunjukkan pada Gambar 3.1.



Gambar 0.1 Metode *Waterfall* (Pressman, 2001)

Metode *waterfall* membagi penelitian menjadi 4 tahap yang saling terkait dan mempengaruhi. Empat tahap tersebut yaitu analisa kebutuhan (*analysis*), desain (*design*), pengkodean (*code*) dan pengujian (*test*) (Pressman, 2001).

3.2.1. Analisa Kebutuhan (*analysis*)

Pada tahap analisa kebutuhan dilakukan pengumpulan informasi mengenai proses yang akan digunakan untuk membangun model kombinasi kriptografi dan steganografi yang meliputi:

a. Pencocokan bit
b. Enkripsi
c. Deksripsi
d. Rekonstruksi pesan
e. Proses *embedding* yang meliputi pencocokan bit dan enkripsi DES
f. Proses ekstraksi yang meliputi dekripsi DES dan rekonstruksi pesan.

Tahap ini diuraikan lebih lanjut pada subbab 3.3 sampai 3.5.

3.2.2. Perancangan Sistem (*design*)

Perancangan adalah proses penterjemahan sistem sesuai algoritma yang digunakan. Hal ini bertujuan agar program yang dibuat sesuai dengan hasil analisa kebutuhan. Hasil perancangan adalah bagan proses, *Data Flow Diagram* (DFD), dan perancangan antarmuka. Tahap ini diuraikan lebih lanjut pada subbab 3.6.

3.2.3. Pengkodean (*code*)

Tahap pengkodean merupakan tahap penterjemahan desain sistem yang telah dibuat ke dalam bentuk perintah-perintah yang dimengerti komputer. Pada penelitian ini dilakukan penulisan kode program sesuai pada langkah desain dengan menggunakan *software* MATLAB R2009a. Pada pengkodean juga dibuat antarmuka sistem untuk mempermudah interaksi antara program dengan *user*.

3.2.4. Pengujian Program (*test*)

Pengujian dilakukan untuk memastikan apakah program yang telah dibuat sesuai dengan desainnya dan semua fungsi dapat dipergunakan dengan baik. Pada penelitian ini pengujian program dilakukan dengan memberikan masukan (*input*) dengan beberapa macam kasus yang mungkin akan ditemukan.

Masukan yang digunakan untuk pengujian pada penelitian ini berupa data teks sebagai pesan rahasia dan *file* citra sebagai media stego. Adapun kriteria data yang diuji sebagai berikut

1) citra dengan warna 100% putih
2) citra dengan warna 100% hitam
3) citra dengan warna hitam putih blok
4) citra dengan warna hitam putih gradasi
5) citra warna “Lenna”, “Pepeer”, “Jet”, Baboon”, “Foto”
6) pesan yang memiliki dengan ukuran lebih besar daripada ukuran citra.

Pengujian citra dengan variasi resolusi 512px, 256px, 128px, dan 64px serta penambahan *noise salt and pepper* dan *gaussian*. Tahap ini diuraikan lebih lanjut pada subbab 3.7.

3.3. Analisa Proses Enkripsi-Dekripsi DES

Berikut ini diberikan sebuah contoh kasus penggunaan algoritma DES untuk pengamanan suatu pesan rahasia. Proses tersebut meliputi a) pembangkitan kunci internal, b) proses enkripsi dan c) proses dekripsi.

Misalkan akan dikirimkan suatu pesan rahasia, sebelum dikirim pesan tersebut terlebih dahulu akan dilakukan proses enkripsi untuk menjaga kerahasiaan isi pesan. Bila diketahui suatu pesan:

Plaintext : komputer
Kunci : 3B3898371520F75E

Keterangan : Kunci menggunakan bilangan Heksa 64 bit (16 digit).

Langkah-langkah enkripsi adalah sebagai berikut.

3.3.1. Pembangkitan Kunci Internal

Karena ada 16 putaran, maka dibutuhkan kunci internal sebanyak 16 buah, yaitu K_1, K_2, …, K_{16}. Kunci internal dibangkitkan dari kunci eksternal yang diberikan oleh pengguna. Langkah-langkah yang dilakukan dalam pembentukan kunci internal:

1) menentukan kunci eksternal dengan panjang 64 bit atau 16 digit bilangan Heksa.
2) mengkonversi plainteks ke dalam bentuk biner yang ditunjukkan pada Tabel 3.1.

Tabel 0.1 Konversi plainteks ke bentuk biner

k	o	m	p	u	t	e	r
01101011	01101111	01101101	01110000	01110101	01110100	01100101	01110010

3) mengkonversi masing-masing bilangan Heksa ke bilangan biner untuk mendapatkan kunci eksternal yang tersusun dari 64 bit. Konversi bilangan Heksa kedalam bentuk biner dapat dilihat pada Tabel 3.2.

Tabel 0.2 Konversi kunci ke bentuk biner

3B	38	98	37	15	20	F7	5E
00111011	00111000	10011000	00110111	00010101	00100000	11110111	01011110

4) melakukan proses pembangkitan kunci-kunci internal DES seperti pada Gambar 2.9. Kunci internal diperoleh dari proses permutasi kunci eksternal menggunakan matriks permutasi kompresi PC-1 (Tabel 3.3) dan dilanjutkan dengan permutasi matriks PC-2 (Tabel 3.6)

Kunci eksternal:

00111011 00111000 10011000 00110111 00010101 00100000 11110111 01011110

Tabel 0.3 Matriks PC-1

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Kunci internal disusun dari perpindahan posisi bit kunci eksternal berdasar matriks PC-1. Bit ke-1 pada kunci internal diperoleh dari posisi bit ke-57 pada kunci eksternal (yaitu 0), bit ke-2 diperoleh dari bit ke-49 pada kunci eksternal (yaitu) dan seterusnya hingga bit

terakhir kunci internal diperoleh dari bit ke-4 pada kunci eksternal (yaitu 1). Setelah dilakukan kompresi dengan permutasi matriks PC-1 diperoleh kunci internal (CD) dengan panjang 56 bit seperti tampak pada Tabel 3.4.

Tabel 0.4 Kunci internal (CD) dengan panjang 56 bit

0	1	0	0	0	1	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	1	1	1	0	1	1	0
0	0	1	0	0	0	0	1	1	1	1	1	1	1

Secara keseluruhan hasil di atas dapat di sajikan secara berurutan mulai dari bit awal sampai bit terakhir sebagai berikut:

0100010 0110000 0001101 0111101 1100100 1110110 0010000 1111111.

5) Kunci internal yang diperoleh dibagi menjadi 2 bagian, kiri (C) dan kanan (D), yang masing-masing panjangnya 28 bit. Bagian C terdiri dari bit pada posisi ke-1 hingga ke-28, sedangkan D terdiri dari bit pada posisi ke-29 hingga ke-56. Susunan biner C dan D adalah sebagai berikut:

C: 0100010 0110000 0001101 0111101

D: 1100100 1110110 0010000 1111111.

Kedua bagian digeser ke kiri (*left shift*) sepanjang 1 atau 2 bit bergantung pada tiap putaran. Aturan pergeseran setiap putaran dapat dilihat pada Tabel 3.5.

Tabel 0.5 Aturan pergeseran setiap putaran

Putaran, *i*	Jumlah pergeseran bit
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Setelah pergeseran bit, dilakukan permutasi (*permuted choice* 2) pada hasil pergeseran bit dari kedua bagian tersebut untuk memperoleh kunci internal pada tiap putaran, dan

dilanjutkan dengan perpindahan lokasi bit menggunakan permutasi matriks PC-2. Matriks PC-2 yang ditunjukkan pada Tabel 3.6.

Tabel 0.6 Matriks PC-2

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Hasil setiap putaran selanjutnya disebut dengan *subkey/*sub kunci, sehingga diperoleh 16 sub kunci internal. Misalkan kunci internal pada kondisi awal dinotasikan dengan CD[0]. Berdasar Tabel 3.5, pada putaran pertama dilakukan pergeseran 1 bit kekiri pada setiap bagian C maupun D, diperoleh hasil CD[1]. Langkah selanjutnya melakukan kompresi dengan matriks permutasi PC-2 diperoleh sub kunci internal 1 dinotasikan KS[1]. Pergeseran kunci internal pada setiap putaran adalah sebagai berikut:

- **Putaran ke-1**

CD[0]:

0100010 0110000 0001101 0111101 1100100 1110110 0010000 1111111

CD[1]:

1000100 1100000 0011010 1111010 1001001 1101100 0100001 1111111

KS[1]:

010111 000000 100001 001100 010101 011000 111101 001111

- **Putaran ke-2**

Menggeser susunan bit CD[1] sebanyak 1 bit ke kiri, diperoleh susunan bit baru CD [2]:

0001001 1000000 0110101 1110101 0010011 1011000 1000011 1111111

Melakukan kompresi susunan bit CD[2] dengan matriks PC-2, diperoleh sub kunci internal 2 dinotasikan KS[2].

KS[2]:

010100 010010 110111 110000 011001 001001 011111 001100

- **Putaran ke-3**

Menggeser susunan bit CD[2] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD [3]:

0100110 0000001 1010111 1010100 1001110 1100010 0001111 1111100

Melakukan kompresi susunan bit CD[2] dengan matriks PC-2, diperoleh sub kunci internal 3 dinotasikan KS[3].

KS[3]:

110101 001110 010010 000101 110110 001011 010011 101111

- **Putaran ke-4**

Menggeser susunan bit CD[3] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[4]:

0011000 0000110 1011110 1010001 0111011 0001000 0111111 1110010

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[4] dengan matriks PC-2, diperoleh sub kunci internal 4 dinotasikan KS[4].

KS[4]:

010100 111000 011100 000110 011011 101101 111010 101001

- **Putaran ke-5**

Menggeser susunan bit CD[4] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[5]:

1100000 0011010 1111010 1000100 1101100 0100001 1111111 1001001

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[5] dengan matriks PC-2, diperoleh sub kunci internal 5 dinotasikan KS[5].

KS[5]:

011010 001001 000010 100111 000110 100111 110101 111011

- **Putaran ke-6**

Menggeser susunan bit CD[5] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[6]:

0000000 1101011 1101010 0010011 0110001 0000111 1111110 0100111

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[6] dengan matriks PC-2, diperoleh sub kunci internal 6 dinotasikan KS[6].

KS[6]:

101100 011000 000001 101110 101011 111101 100100 110000

- **Putaran ke-7**

Menggeser susunan bit CD[6] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[7]:

0000011 0101111 0101000 1001100 1000100 0011111 1111001 0011101

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[7] dengan matriks PC-2, diperoleh sub kunci internal 7 dinotasikan KS[7].

KS[7]:

101000 000100 001010 110010 110000 010110 111101 110010

- **Putaran ke-8**

Menggeser susunan bit CD[7] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[]:

0001101 0111101 0100010 0110000 0010000 1111111 1100100 1110110

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[8] dengan matriks PC-2, diperoleh sub kunci internal 8 dinotasikan KS[8].

KS[8]:

101101 000001 101100 110100 111111 011000 101000 011100

- **Putaran ke-9**

Menggeser susunan bit CD[8] sebanyak 1 bit ke kiri, diperoleh susunan bit baru CD[9]:

0011010 1111010 1000100 1100000 0100001 1111111 1001001 1101100

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[9] dengan matriks PC-2, diperoleh sub kunci internal 9 dinotasikan KS[9].

KS[9]:

001000 101101 110101 000010 100100 111000 011001 111100

- **Putaran ke-10**

Menggeser susunan bit CD[9] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[10]:

1101011 1101010 0010011 0000000 0000111 1111110 0100111 0110001

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[10] dengan matriks PC-2, diperoleh sub kunci internal 10 dinotasikan KS[10].

KS[10]:

011010 000110 000101 010111 110110 011011 111110 000100

- **Putaran ke-11**

Menggeser susunan bit CD[10] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[11]:

0101111 0101000 1001100 0000011 0011111 1111001 0011101 1000100

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[11] dengan matriks PC-2, diperoleh sub kunci internal 11 dinotasikan KS[11].

KS[11]:

001001 011100 010100 011001 001110 000110 011010 111101

- **Putaran ke-12**

Menggeser susunan bit CD[11] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[12]:

0111101 0100010 0110000 0001101 1111111 1100100 1110110 0010000

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[12] dengan matriks PC-2, diperoleh sub kunci internal 12 dinotasikan KS[12].

KS[12]:

010001 110000 000110 110011 011110 110111 100010 000111

- **Putaran ke-13**

Menggeser susunan bit CD[12] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[13]:

1110101 0001001 1000000 0110101 1111111 0010011 1011000 1000011

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[13] dengan matriks PC-2, diperoleh sub kunci internal 13 dinotasikan KS[13].

KS[13]:

101111 111000 100010 010001 101001 100110 000110 111011

- **Putaran ke-14**

Menggeser susunan bit CD[13] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[14]:

1010100 0100110 0000001 1010111 1111100 1001110 1100010 0001111

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[14] dengan matriks PC-2, diperoleh sub kunci internal 14 dinotasikan KS[14].

KS[14]:

000111 110010 001010 001010 101001 110011 101101 000111

- **Putaran ke-15**

Menggeser susunan bit CD[14] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[15]:

1010001 0011000 0000110 1011110 1110010 0111011 0001000 0111111

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[15] dengan matriks PC-2, diperoleh sub kunci internal 15 dinotasikan KS[15].

KS[15]:

001110 100001 010010 011100 111101 101000 001111 110010

- **Putaran ke-16**

Menggeser susunan bit CD[15] sebanyak 2 bit ke kiri, diperoleh susunan bit baru CD[16]:

0100010 0110000 0001101 0111101 1100100 1110110 0010000 1111111

Langkah selanjutnya, melakukan kompresi pada susunan bit CD[16] dengan matriks PC-2, diperoleh sub kunci internal 16 dinotasikan KS[16].

KS[16]:

000100 010111 110010 000001 110101 111110 000101 001110

3.3.2. Proses Enkripsi

Proses enkripsi terhadap blok plainteks dilakukan setelah permutasi awal. Setiap blok plainteks mengalami 16 kali putaran enkripsi (Gambar 2.7). Proses substitusi dilakukan dengan menggunakan delapan buah kotak-S (*S-box*), S_1 sampai S_8. Setiap kotak-S menerima masukan 6 bit dan menghasilkan keluaran 4 bit. Kelompok 6-bit pertama menggunakan S_1, kelompok 6-bit kedua menggunakan S_2, dan seterusnya.

Diketahui plainteks:

k	o	m	p	u	t	e	r
01101011	01101111	01101101	01110000	01110101	01110100	01100101	01110010

Plainteks diatas diacak dengan menggunakan aturan matriks permutasi awal yang ditunjukkan pada Tabel 3.7

Tabel 0.7 Matriks permutasi awal

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Hasil plainteks dengan permutasi awal dinotasikan dengan LR yaitu:

Tabel 0.8 Plainteks setelah dilakukan permutasi awal

1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0
0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1

Hasil plainteks (LR) yang diperoleh dibagi menjadi 2 bagian, kiri (L) dan kanan (L), yang masing-masing panjangnya 32 bit. Plainteks L terdiri dari bit pada posisi ke-1 hingga ke-32, plainteks R terdiri dari bit pada posisi ke-33 hingga ke-64. Kondisi awal plainteks sebelum proses iterasi adalah sebagai berikut:

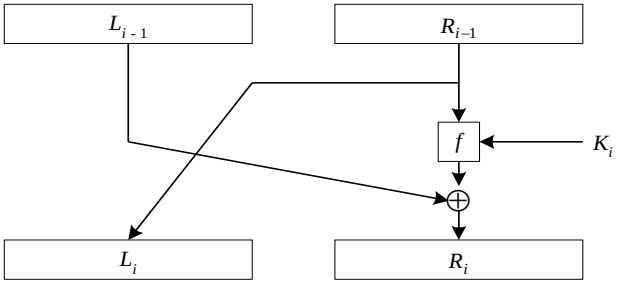
L[0] : 11111111 10111000 01110110 01010111

R[0] : 00000000 11111111 00000111 10000011

Proses enkripsi dilakukan dalam 16 putaran sebagai berikut:

- **Putaran 1**

Satu putaran DES merupakan model jaringan *Feistel* seperti tampak pada Gambar 3.2.



Gambar 0.2 Model jaringan *Feistel*

Langkah pertama dengan memperluas blok R yang panjangnya 32-bit menjadi 48-bit dengan fungsi ekspansi. Fungsi ekspansi direalisasikan dengan matriks permutasi ekspansi yang ditunjukkan pada Tabel 3.9.

Tabel 0.9 Matriks permutasi ekspansi

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17

16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Hasil ekspansi R dinotasikan dalam E.

E : 100000 000001 011111 111110 100000 001111 110000 000110

Selanjutnya hasil ekspansi (E) yang panjangnya 48-bit di-XOR-kan dengan sub kunci internal KS[1] yang panjangnya 48-bit, menghasikan vektor A dengan panjang 48-bit.

KS[1] : 010111 000000 100001 001100 010101 011000 111101 001111

E xor KS (A) : 110111 000001 111110 110010 110101 010111 001101 001001

Hasil vektor A kemudian disubtitusi menggunakan matriks S-*box*. Kotak S-*box* memetakan 6 bit masukan menjadi 4 bit keluaran. Kelompok 6 bit pertama menggunakan kotak S_1 (Tabel 2.8), 6 bit ke-2 menggunakan S_2 dan seterusnya hingga 6 bit ke-8 menggunakan S_8. Hasil keluaran proses subtitusi tersebut secara rinci proses subtitusi sebagai berikut.

1) Kelompok 6 bit pertama yaitu 110111. Langkah pertama adalah menentukan nomor baris tabel yang diperoleh dari nilai bit pertama (b_1) dan nilai bit ke 6 (b_6), yaitu b1: 1 dan b6: 1. Sehingga diperoleh nomor baris tabel 11 (biner) atau 3 (desimal). Artinya baris ke-3. Langkah kedua menentukan nomor kolom tabel yang diperoleh dari nilai bit ke-2 (b_2) sampai bit ke-5 (b_5), yaitu b_2: 1, b_3: 0, b_4: 1 dan b_5: 1. Secara keseluruhan nomor kolom yaitu 1011 (biner) atau 11 (desimal). Artinya nomor kolom tabel terletak pada kolom ke 11. Jadi subtitusi untuk 110111 adalah *entry* baris ke-3 kolom ke-11 pada kotak S_1 yaitu 14 (desimal) dan jika dinyatakan dalam bentuk biner menghasilkan 1110.

Tabel 0.10 Proses subtitusi S-*box* 1

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

2) Kelompok 6 bit kedua yaitu 000001. Seperti pada langkah sebelumnya diperoleh nomor baris tabel yaitu 01 (1 desimal) dan nomor kolom 0000 (0 desimal). Jadi subtitusi untuk 000001 adalah baris ke-1 dan kolom ke-0 pada S_2 yaitu 3 desimal atau 0011 dalam biner.

Tabel 0.11 Proses subtitusi S-*box* 2

S-*box* 2

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

3) Kelompok 6 bit ketiga yaitu 111110. Diperoleh nomor baris tabel yaitu 10 (2 desimal) dan nomor kolom tabel 1111 (15 desimal). Jadi subtitusi untuk 111110 adalah baris ke-2 dan kolom ke-15 pada S_3 yaitu 7 desimal atau 011 dalam bentuk biner.

Tabel 0.12 Proses subtitusi S-*box* 3

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

4) Kelompok 6 bit keempat yaitu 110010. Diperoleh nomor baris tabel yaitu 10 (2 desimal) dan nomor kolom tabel 1001 (9 desimal). Jadi subtitusi untuk 110010 adalah baris ke-2 dan kolom ke-9 pada S_4 yaitu 1 desimal atau 0001 dalam bentuk biner.

Tabel 0.13 Proses subtitusi S-*box* 4

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

5) Kelompok 6 bit kelima yaitu 110101. Diperoleh nomor baris tabel yaitu 11 (3 desimal) dan nomor kolom tabel 1010 (10 desimal). Jadi subtitusi untuk 110101 adalah baris ke-3 dan kolom ke-10 pada S_5 yaitu 0 desimal atau 0000 dalam bentuk biner.

Tabel 0.14 Proses subtitusi S-*box* 5

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

6) Kelompok 6 bit keenam yaitu 01011. Diperoleh nomor baris tabel yaitu 01 (1 desimal) dan nomor kolom tabel 1011 (11 desimal). Jadi subtitusi untuk 010111 adalah baris ke-1 dan kolom ke-11 pada kotak S_6 yaitu 14 desimal atau 1110 dalam bentuk biner.

Tabel 0.15 Proses subtitusi S-*box* 6

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

7) Kelompok 6 bit ketujuh yaitu 001101. Diperoleh nomor baris tabel yaitu 01 (1 desimal) dan nomor kolom tabel 0110 (6 desimal). Jadi subtitusi untuk 001101 adalah baris ke-1 dan kolom ke-6 pada kotak S_7 yaitu 1 desimal atau 0001 dalam bentuk biner.

Tabel 0.16 Proses subtitusi S-*box* 7

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

8) Kelompok 6 bit kedelapan yaitu 001001. Diperoleh nomor baris tabel yaitu 01 (1 desimal) dan nomor kolom tabel 0100 (4 desimal). Jadi subtitusi untuk 001001 adalah baris ke-1 dan kolom ke-4 pada kotak S_8 yaitu 10 desimal atau 1010 dalam bentuk biner.

Tabel 0.17 Proses subtitusi S-*box* 8

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

Secara keseluruhan hasil keluaran proses subtitusi oleh matriks S-*box* adalah yaitu vektor B dengan panjang 48 bit.

S-*box* (vektor B) : 1110 0011 0111 0001 0000 1110 0001 1010

Vektor *B* menjadi masukan untuk proses permutasi. Tujuan permutasi adalah untuk mengacak hasil proses substitusi kotak-S. Permutasi dilakukan dengan menggunakan matriks permutasi *P* (*P-box*) yang ditunjukkan pada Tabel 3.18.

Tabel 0.18 Matriks permutasi P (P-*box*)

16	7	20	21	29	12	28	17	1	15	23	26	5	8	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Hasil permutasi vektor B dinyatakan dengan P(B) sebagai berikut:

P (B) : 11011110 10100011 11000010 00001100.

Akhirnya bit-bit P(B) di-XOR-kan dengan L_{i-1} untuk mendapatkan R_i, secara matematis dapat ditulis $R_i = L_{i-1} \oplus P(B)$.

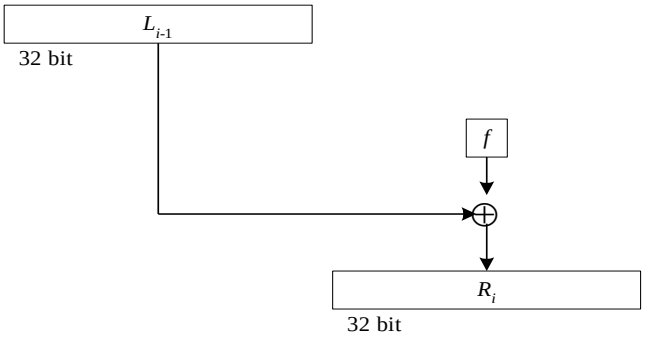
Nilai bit L_{i-1} adalah

L[0] : 11111111 10111000 01110110 01010111

Setelah di XOR-kan, hasil R_i yang diperoleh yaitu:

R[i] : 00100001 00011011 10110100 01011011.

Skema perolehan Ri ditunjukkan pada Gambar 3.3.

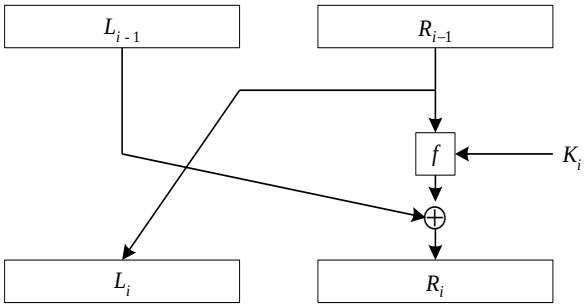


Gambar 0.3 Proses perolehan L_i

Selanjutnya R_{i-1} yaitu R[0] : 00000000 11111111 00000111 10000011 menjadi masukan untuk L_i, sehingga diperoleh:

L[i] : 00000000 11111111 00000111 10000011.

Skema perolehan Ri dan Li ditunjukkan pada Gambar 3.4.



Gambar 0.4 Proses perolehan L_i dan R_i

Jadi hasil akhir keluaran putaran pertama yaitu:

L[i] : 00000000 11111111 00000111 10000011

R[i] : 00100001 00011011 10110100 01011011.

- **Putaran 2**

Langkah pada putaran 2 seperti pada putaran 1, diperoleh hasil ekspansi (E):

E : 100100 000010 100011 110111 110110 101000 001011 110110

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[2] yang panjangnya 48-bit, menghasikan vektor A. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[2] : 010100 010010 110111 110000 011001 001001 011111 001100

E xor KS (A) : 110000 010000 010100 000111 101111 100001 010100 111010

S-*box* (B) : 1111 1001 1100 0101 1101 0100 1001 0011

P (B) : 10100011 10001111 11011011 00001011

L[i] : 00100001 00011011 10110100 01011011

R[i] : 10100011 01110000 11011100 10001000

- **Putaran 3**

Langkah pada putaran ketiga sama seperti pada putaran kedua dengan ekspansi (E) sebagai berikut:

E : 010100 000110 101110 100001 011011 111001 010001 010001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[3] yang panjangnya 48-bit, menghasikan vektor A. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[3] : 110101 001110 010010 000101 110110 001011 010011 101111

E xor KS : 100001 001000 111100 100100 101101 110010 000010 111110

S-*box* : 1111 0110 1110 1001 0010 0000 1011 1000

P : 11001010 10000001 10000111 11010111

L[i] : 10100011 01110000 11011100 10001000

R[i] : 11101011 10011010 00110011 10001100

- **Putaran 4**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011101 010111 110011 110100 000110 100111 110001 011001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[4] yang panjangnya 48-bit, menghasikan vektor A. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[4] : 010100 111000 011100 000110 011011 101101 111010 101001

E xor KS : 001001 101111 101111 110010 011101 001010 001011 110000

S-*box* : 1110 0010 0111 0001 1000 0010 1001 0000

P : 11000111 10100001 10000010 00000101

L[i] : 11101011 10011010 00110011 10001100

R[i] : 01100100 11010001 01011110 10001101

- **Putaran 5**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 101100 001001 011010 100010 101011 111101 010001 011010

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[5] yang panjangnya 48-bit, menghasikan vektor A. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[5] : 011010 001001 000010 100111 000110 100111 110101 111011

E xor KS : 110110 000000 011000 000101 101101 011010 100100 100001

S-*box* : 0111 1111 1011 1011 0010 0111 1011 0010

P : 11000110 01101010 11100111 11011111

L[i] : 01100100 11010001 01011110 10001101

R[i] : 00101101 11110000 11010100 01010011

- **Putaran 6**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 100101 011011 111110 100001 011010 101000 001010 100110

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[6] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[6] : 101100 011000 000001 101110 101011 111101 100100 110000
E xor KS : 001001 000011 111111 001111 110001 010101 101110 010110
S-*box* : 1110 1101 1100 0011 0110 1101 1110 1110
P(B) : 10011000 11011111 11100111 10111001
L[i] : 00101101 11110000 11010100 01010011
R[i] : 11111100 00001110 10111001 00110100

- **Putaran 7**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011111 111000 000001 011101 010111 110010 100110 101001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[7] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[7] : 101000 000100 001010 110010 110000 010110 111101 110010
E xor KS : 110111 111100 001011 101111 100111 100100 011011 011011
S-*box* : 1110 0010 0100 1000 0111 1111 1111 1110
P(B) : 01111010 10110111 10100110 11101001
L[i] : 11111100 00001110 10111001 00110100
R[i] : 01010111 01000111 01110010 10111010.

- **Putaran 8**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 001010 101110 101000 001110 101110 100101 010111 110100

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[8] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[8] : 101101 000001 101100 110100 111111 011000 101000 011100
E xor KS : 100111 101111 000100 111010 010001 111101 111111 101000
S-*box* : 0010 0010 1001 0010 0101 1000 1100 1001
P(B) : 01111100 01010100 00001011 00000001
L[i] : 01010111 01000111 01110010 10111010
R[i] : 10000000 01011010 10110010 00110101

- **Putaran 9**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110000 000000 001011 110101 010110 100100 000110 101011

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[9] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[9] : 001000 101101 110101 000010 100100 111000 011001 111100
E xor KS : 111000 101101 111110 110111 110010 011100 011111 010111
S-*box* : 0011 0100 0111 1011 1001 0101 0110 1011
P : 10101101 01010011 00101110 01011110
L[i] : 10000000 01011010 10110010 00110101
R[i] : 11111010 00010100 01011100 11100100

- **Putaran 10**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011111 110100 000010 101000 001011 111001 011100 001001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[10] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[10] : 011010 000110 000101 010111 110110 011011 111110 000100

E xor KS : 000101 110010 000111 111111 111101 100010 100010 001101

S-*box* : 0111 1000 1001 1110 0101 1110 0100 0111

P : 00110100 01111110 10011011 01101010

L[i] : 11111010 00010100 01011100 11100100

R[i] : 10110100 00100100 00101001 01011111

- **Putaran 11**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110110 101000 000100 001000 000101 010010 101011 111111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[11] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[11] : 001001 011100 010100 011001 001110 000110 011010 111101

E xor KS : 111111 110100 010000 010001 001011 010100 110001 000010

S-*box* : 1101 1100 0001 0100 0111 0011 1001 0010

P : 00100110 10101110 10110000 10010011

L[i] : 10110100 00100100 00101001 01011111

R[i] : 11011100 10111010 11101100 01110111

- **Putaran 12**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 111011 111001 010111 110101 011101 011000 001110 101111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[12] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[12] : 010001 110000 000110 110011 011110 110111 100010 000111

E xor KS : 101010 001001 010001 000110 000011 101111 101100 101000

S-*box* : 0110 1111 0010 0011 1011 1010 0111 1001

P : 11111011 01111000 11001110 10010100

L[i] : 11011100 10111010 11101100 01110111

R[i] : 01001111 01011100 11100111 11001011

- **Putaran 13**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 101001 011110 101011 111001 011100 001111 111001 010110

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[13] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[13] : 101111 111000 100010 010001 101001 100110 000110 111011

E xor KS : 000110 100110 001001 101000 110101 101001 111111 101101

S-*box* : 0001 1011 0011 1100 0000 1001 1100 1000

P : 01011100 00011000 01110000 01000111

L[i] : 01001111 01011100 11100111 11001011

R[i] : 10000000 10100010 10011100 00110000

- **Putaran 14**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 010000 000001 010100 000101 010011 111000 000110 100001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[14] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[14] : 000111 110010 001010 001010 101001 110011 101101 000111

E xor KS : 010111 110011 011110 001111 111010 001011 101011 100110

S-*box* : 1011 0110 1000 0011 0011 1100 0100 0001

P : 11110000 11010000 00001011 10011010

L[i] : 10000000 10100010 10011100 00110000

R[i] : 10111111 10001100 11101100 01010001

- **Putaran 15**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110111 111111 110001 011001 011101 011000 001010 100011

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[15] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[15] : 001110 100001 010010 011100 111101 101000 001111 110010

E xor KS : 111001 011110 100011 000101 100000 110000 000101 010001

S-*box* : 1010 1010 1010 1011 0100 0111 1011 1100

P : 11001010 11101100 00100111 01101101

L[i] : 10111111 10001100 11101100 01010001

R[i] : 01001010 01001110 10111011 01011101

- **Putaran 16**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 101001 010100 001001 011101 010111 110110 101011 111010

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[16] yang panjangnya 48-bit, menghasikan vektor A yang dinyatakan E xor KS. Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran vektor B, yaitu S-*box* (B). Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran pada setiap langkah adalah sebagai berikut:

KS[16] : 000100 010111 110010 000001 110101 111110 000101 001110

E xor KS : 101101 000011 111011 011100 100010 001000 101110 110100

S-*box* : 0001 1101 0101 0100 0010 1001 1110 1010

P : 00011100 00011011 01110100 10010011

L[i] : 01001010 01001110 10111011 01011101

R[i] : 10100011 10010111 10011000 11000010

Hasil gabungan blok kiri Li dan blok kanan Ri yaitu LR[16]

LR[16] :

10100011 10010111 10011000 11000010 01001010 01001110 10111011 01011101

Langkah terakhir yaitu melakukan permutasi akhir setelah 16 kali putaran terhadap gabungan blok kiri dan blok kanan. Proses permutasi menggunakan matriks permutasi awal balikan (*inverse initial permutation* atau IP^{-1}) yang ditunjukkan pada Tabel 3.19.

Tabel 0.19 Matriks IP^{-1}

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Setelah dilakukan permutasi terakhir maka akan diperoleh chiperteks yang ditunjukkan pada Tabel 3.20.

Tabel 0.20 Hasil chiperteks

01011010	11111001	00110010	10101110	00011110	01001000	10100011	01011101
5A	F9	32	AE	1E	48	A3	5D

Jadi bila diketahui :

Plainteks : komputer

Kunci : 3B3898371520F75E

Chiperteks yang dihasilkan adalah : 5AF932AE1E48A35D.

3.3.3. Proses Dekripsi

Proses dekripsi terhadap chiperteks merupakan kebalikan dari proses enkripsi. DES menggunakan algoritma yang sama untuk proses enkripsi dan dekripsi. Jika pada proses enkripsi urutan kunci internal yang digunakan adalah K_1, K_2, …, K_{16}, maka pada proses dekripsi urutan kunci yang digunakan adalah K_{16}, K_{15}, …, K_1.

Langkah untuk melakukan dekripsi pesan tersebut adalah:

Mengkonversi chiperteks menjadi bentuk biner untuk mendapatkan bit chiperteks.

Tabel 0.21 Konversi chiperteks menjadi bentuk biner

5A	F9	32	AE	1E	48	A3	5D
01011010	11111001	00110010	10101110	00011110	01001000	10100011	01011101

Melakukan pembangkitan kunci internal DES sama seperti pada langkah sebelumnya. Kunci internal diperoleh dari permutasi kunci eksternal menggunakan matriks permutasi kompresi PC-1 dan dilanjutkan dengan permutasi matriks PC-2. Proses tersebut sama seperti pada pembangkitan kunci internal proses enkripsi (subbab 3.3.1).

Melakukan proses dekripsi sama seperti proses enkripsi, namun dengan penggunaan urutan kunci internal (KS) yang berbeda (kebalikan). Jika proses enkripsi urutan kunci internal dimulai dari KS[1], KS[2], ..., KS[16], maka pada proses dekripsi unrutan kunci internal sebaliknya, yaitu mulai KS[16], KS[15], ..., KS[1].

Proses dekripsi secara keseluruhan terjadi dalam 16 putaran sebagai berikut:

- **Putaran 1**

Chiperteks yang dihasilkan kemudian dilakukan permutasi dengan matriks kebalikan IP^{-1}. Pra-keluaran dari *deciphering* adalah adalah (L_0, R_0). Hasil (LR) yang diperoleh dibagi menjadi 2 bagian, kiri (L) dan kanan (L) yang masing-masing panjangnya 32 bit. Bagian L terdiri dari bit pada posisi ke-1 hingga ke-32, R terdiri dari bit pada posisi ke-33 hingga ke-64. Tahap selanjutnya yaitu melakukan fungsi ekspansi seperti pada proses enkripsi. Diperoleh hasil ekspansi yang dinotasikan dalam E.

E : 101001 010100 001001 011101 010111 110110 101011 111010

KS[16] : 000100 010111 110010 000001 110101 111110 000101 001110

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[16] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 101101 000011 111011 011100 100010 001000 101110 110100

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0001 1101 0101 0100 0010 1001 1110 1010

P(B) : 00011100 00011011 01110100 10010011

L[i] : 01001010 01001110 10111011 01011101

R[i] : 10111111 10001100 11101100 01010001.

- **Putaran 2**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110111 111111 110001 011001 011101 011000 001010 100011

KS : 001110 100001 010010 011100 111101 101000 001111 110010

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[15] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 111001 011110 100011 000101 100000 110000 000101 010001

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1010 1010 1010 1011 0100 0111 1011 1100

P : 11001010 11101100 00100111 01101101

L[i] : 10111111 10001100 11101100 01010001

R[i] : 10000000 10100010 10011100 00110000.

- **Putaran 3**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 010000 000001 010100 000101 010011 111000 000110 100001

KS : 000111 110010 001010 001010 101001 110011 101101 000111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[14] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 010111 110011 011110 001111 111010 001011 101011 100110

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1011 0110 1000 0011 0011 1100 0100 0001

P : 11110000 11010000 00001011 10011010

L[i] : 10000000 10100010 10011100 00110000

R[i] : 01001111 01011100 11100111 11001011.

- **Putaran 4**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 101001 011110 101011 111001 011100 001111 111001 010110

KS : 101111 111000 100010 010001 101001 100110 000110 111011

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[13] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 000110 100110 001001 101000 110101 101001 111111 101101

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0001 1011 0011 1100 0000 1001 1100 1000

P : 01011100 00011000 01110000 01000111

L[i] : 01001111 01011100 11100111 11001011

R[i] : 11011100 10111010 11101100 01110111

- **Putaran 5**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 111011 111001 010111 110101 011101 011000 001110 101111

KS : 010001 110000 000110 110011 011110 110111 100010 000111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[12] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 101010 001001 010001 000110 000011 101111 101100 101000

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0110 1111 0010 0011 1011 1010 0111 1001

P : 11111011 01111000 11001110 10010100

L[i] : 11011100 10111010 11101100 01110111

R[i] : 10110100 00100100 00101001 01011111.

- **Putaran 6**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110110 101000 000100 001000 000101 010010 101011 111111

KS : 001001 011100 010100 011001 001110 000110 011010 111101

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[11] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 111111 110100 010000 010001 001011 010100 110001 000010

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1101 1100 0001 0100 0111 0011 1001 0010

P : 00100110 10101110 10110000 10010011

L[i] : 10110100 00100100 00101001 01011111

R[i] : 11111010 00010100 01011100 11100100.

- **Putaran 7**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011111 110100 000010 101000 001011 111001 011100 001001

KS : 011010 000110 000101 010111 110110 011011 111110 000100

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[10] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 000101 110010 000111 111111 111101 100010 100010 001101

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0111 1000 1001 1110 0101 1110 0100 0111

P : 00110100 01111110 10011011 01101010

L[i] : 11111010 00010100 01011100 11100100

R[i] : 10000000 01011010 10110010 00110101.

- **Putaran 8**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 110000 000000 001011 110101 010110 100100 000110 101011

KS : 001000 101101 110101 000010 100100 111000 011001 111100

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[9] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 111000 101101 111110 110111 110010 011100 011111 010111

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0011 0100 0111 1011 1001 0101 0110 1011

P : 10101101 01010011 00101110 01011110

L[i] : 10000000 01011010 10110010 00110101

R[i] : 01010111 01000111 01110010 10111010.

- **Putaran 9**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 001010 101110 101000 001110 101110 100101 010111 110100

KS : 101101 000001 101100 110100 111111 011000 101000 011100

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[8] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 100111 101111 000100 111010 010001 111101 111111 101000

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0010 0010 1001 0010 0101 1000 1100 1001

P : 01111100 01010100 00001011 00000001

L[i] : 01010111 01000111 01110010 10111010

R[i] : 11111100 00001110 10111001 00110100.

- **Putaran 10**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011111 111000 000001 011101 010111 110010 100110 101001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[7] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

KS : 101000 000100 001010 110010 110000 010110 111101 110010

E xor KS : 110111 111100 001011 101111 100111 100100 011011 011011

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1110 0010 0100 1000 0111 1111 1111 1110

P : 01111010 10110111 10100110 11101001

L[i] : 11111100 00001110 10111001 00110100

R[i] : 00101101 11110000 11010100 01010011.

- **Putaran 11**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 100101 011011 111110 100001 011010 101000 001010 100110

KS : 101100 011000 000001 101110 101011 111101 100100 110000

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[6] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 001001 000011 111111 001111 110001 010101 101110 010110

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1110 1101 1100 0011 0110 1101 1110 1110

P : 10011000 11011111 11100111 10111001

L[i] : 00101101 11110000 11010100 01010011

R[i] : 01100100 11010001 01011110 10001101.

- **Putaran 12**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 101100 001001 011010 100010 101011 111101 010001 011010

KS : 011010 001001 000010 100111 000110 100111 110101 111011

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[5] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 110110 000000 011000 000101 101101 011010 100100 100001

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 0111 1111 1011 1011 0010 0111 1011 0010

P : 11000110 01101010 11100111 11011111

L[i] : 01100100 11010001 01011110 10001101

R[i] : 11101011 10011010 00110011 10001100.

- **Putaran 13**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 011101 010111 110011 110100 000110 100111 110001 011001

KS : 010100 111000 011100 000110 011011 101101 111010 101001

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[4] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut

E xor KS : 001001 101111 101111 110010 011101 001010 001011 110000

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1110 0010 0111 0001 1000 0010 1001 0000

P : 11000111 10100001 10000010 00000101

L[i] : 11101011 10011010 00110011 10001100

R[i] : 10100011 01110000 11011100 10001000.

- **Putaran 14**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 010100 000110 101110 100001 011011 111001 010001 010001

KS : 110101 001110 010010 000101 110110 001011 010011 101111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[3] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 100001 001000 111100 100100 101101 110010 000010 111110

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1111 0110 1110 1001 0010 0000 1011 1000

P : 11001010 10000001 10000111 11010111

L[i] : 10100011 01110000 11011100 10001000

R[i] : 00100001 00011011 10110100 01011011.

- **Putaran 15**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 100100 000010 100011 110111 110110 101000 001011 110110

KS : 010100 010010 110111 110000 011001 001001 011111 001100

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[2] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 110000 010000 010100 000111 101111 100001 010100 111010

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1111 1001 1100 0101 1101 0100 1001 0011

P : 10100011 10001111 11011011 00001011

L[i] : 00100001 00011011 10110100 01011011

R[i] : 00000000 11111111 00000111 10000011

- **Putaran 16**

Langkah pada putaran ini sama seperti pada putaran sebelumnya. Diperoleh hasil ekspansi (E) sebagai berikut:

E : 100000 000001 011111 111110 100000 001111 110000 000110

KS : 010111 000000 100001 001100 010101 011000 111101 001111

Selanjutnya hasil ekspansi (E) di-XOR-kan dengan sub kunci internal KS[1] yang panjangnya 48-bit, menghasilkan vektor A sebagai berikut:

E xor KS : 110111 000001 111110 110010 110101 010111 001101 001001

Vektor A kemudian disubtitusi menggunakan matriks S-*box* menghasilkan keluaran S-*box*. Hasil S-*box* kemudian menjadi masukan untuk permutasi P, diperoleh P(B) dan hasil terakhir yaitu L[i] dan R[i]. Hasil keluaran S-*box*, P(B), L[i] dan R[i] adalah sebagai berikut:

S-*box* : 1110 0011 0111 0001 0000 1110 0001 1010

P : 11011110 10100011 11000010 00001100

L[i] : 00000000 11111111 00000111 10000011

R[i] : 11111111 10111000 01110110 01010111

Hasil gabungan blok kiri Li dan blok kanan Ri yaitu LR[16]

LR[16] :

11111111	10111000	01110110	01010111	00000000	11111111	00000111	10000011

Langkah terakhir yaitu melakukan permutasi dengan matriks awal dan mengkonversi dari bentuk biner menjadi karakter. Sehingga menghasilkan plainteks seperti tampak pada Tabel 3.22.

Tabel 0.22 Plainteks hasil dekripsi

01101011	01101111	01101101	01110000	01110101	01110100	01100101	01110010
k	o	m	p	u	t	e	r

Jadi bila diketahui:

Ciphertext : 5AF932AE1E48A35D

Kunci : 3B3898371520F75E

Plainteks yang dihasilkan adalah: komputer.

3.4. Analisa Proses Steganografi

Berikut ini diberikan sebuah contoh kasus steganografi dengan masukan berupa pesan dan citra. Proses steganografi meliputi 2 tahap yaitu pencocokan bit dan rekonstruksi.

3.4.1. Pencocokan Bit

Masukan pada tahap ini adalah pesan dan citra. Langkah-langkah yang dilakukan pada pencocokan bit adalah:

1) Mengkonversi pesan dan citra dalam bentuk biner
2) Mengambil nilai MSB citra
3) Melakukan pencocokan pesan pada MSB citra. Jika bit pesan terdapat pada MSB citra, maka dilanjutkan dengan menyimpan posisi indeks bit. Penyimpanan indeks terdiri dari posisi indeks bit awal (*start*) dan posisi indeks bit akhir (*end*). Jika proses pencocokan tidak terjadi, dilanjutkan proses d) sebagai berikut
4) Membagi pesan menjadi dua bagian sama panjang kiri (L[i]) dan kanan (R[i])
5) Mengulangi langkah yang sama seperti pada nomor b), dengan L[i] dan R[i] sebagai masukan. Jika semua bit pesan terdapat pada citra, maka pencocokan selesai dan dilanjutkan proses f). Jika tidak, mengulangi langkah c) dengan L[i] dan R[i] sebagai masukkan hingga proses ke-i.
6) Menyimpan semua indeks bit hasil pencocokan
7) Keluaran berupa vektor yang memuat susunan indeks posisi bit.

   Sebagai contoh, misalkan diketahui bit pesan dan bit citra sebagai berikut:

   Pesan (P) : 10110111

   Citra (C) : 100100011010110101010011

Langkah pencocokan lokasi bit pesan (dinotasikan P) pada citra (dinotasikan C) sebagai berikut:

1) Mencocokan P: 10110111 dengan panjang 8 bit pada C: 100100011010110101010011.
2) Karena P tidak terdapat pada C maka P dipecah menjadi dua bagian kiri (L) dan kanan (R), yaitu L[1]: 1011 dan R[1]: 0111.
3) Mengulangi seperti langkah 1) dengan L[1] dan R[1] sebagai masukkan
   a. Kasus L[1]:

      Mencocokan bit L[1]: 1011 pada C. Diperoleh kesamaan pada lokasi index ke 11 hingga 14, yaitu 10010001101<u>1011</u>0101010011.

      Menulis vektor lokasi bit yang sama yaitu “11 14”.

   b. Kasus R[1]:

      Mencocokan bit R[1]: 0111 pada C.

      Karena tidak diperoleh kesamaan pada C, sehingga P_R dipecah menjadi 2 bagian sama panjang kiri dan kanan, yaitu L[2]: 01 dan R[2]: 11.

4) Mengulangi seperti langkah 1) dengan L[2] dan R[2] sebagai masukkan.
   a. Kasus L[2]:

      Mencocokan bit kiri 01 pada C. Diperoleh kesamaan pada lokasi index ke 3 hingga 4, yaitu 10<u>01</u>00011010110101010011. Menulis vektor lokasi bit yang sama yaitu “3 4”.
   b. Kasus R[2]:

      Mencocokan bit kanan 11 pada C. Diperoleh kesamaan pada lokasi index ke 8 hingga 9, yaitu 1001000<u>11</u>010110101010011. Menulis vektor lokasi bit yang sama yaitu “8 9”.
5) Karena semua posisi bit sudah ditemukan, maka proses pencocokan selesai dan dilanjutkan langkah 6.
6) Menggabungkan semua solusi yang diperoleh pada proses 3a), 4a, dan 4b) yaitu “11 14”, “3 4”, dan “8 9”. Jadi solusi totalnya adalah 11 14 3 4 8 9.
7) Keluaran berupa vektor yang memuat posisi pada indeks bit 11 14 3 4 8 9.

3.4.2. Rekonstruksi Pesan

Rekonstruksi bertujuan untuk mengembalikan pesan menjadi bentuk semula. Masukan pada tahap ini terdiri dari indeks bit dan citra. Proses yang dilakukan yaitu dengan mengambil susunan bit citra berdasar indeks bit. Hasil keluaran berupa susunan bit pesan. Langkah-langkah yang dilakukan pada proses rekonstruksi adalah:

a) Mengkonversi citra dalam bentuk biner dan mengambil bit MSB citra.
b) Membaca setiap dua indeks isi vektor. Indeks pertama merupakan posisi awal bit (*start*) dan indeks kedua merupakan posisi akhir bit (*end*),
c) Mengambil nilai bit citra berdasarkan langkah b),
d) Mengulangi proses b) dan c) sampai posisi indeks terakhir.
e) Susunan bit yang terbentukakan menghasilkan keluaran berupa susunan bit pesan.

Sebagai contoh, misalkan diketahui vektor dan citra sebagai berikut:

Vektor : 11 14 3 4 8 9

Citra : 1001000110101101010010011

Langkah ekstraksi yaitu dengan mengambil nilai bit citra berdasarkan lokasi vektor. Adapun langkah pencocokan yaitu sebagai berikut:

1. Membaca dua angka pertama isi vektor, pada kasus tersebut yaitu 11 14. Artinya mengambil lokasi index ke 11 hingga 14 pada bit citra 10010001101<u>0110</u>101010011, diperoleh 1011.
2. Mengulangi langkah seperti nomor 1 hingga vektor terakhir.

Dua angka kedua:

Pada kasus ini dua angka kedua yaitu 3 4. Artinya mengambil lokasi bit citra pada posisi index ke 3 sampai 4, diperoleh 01.

Dua angka ketiga:

Mengambil nilai bit citra dengan lokasi vektor 8 9, yaitu mengambil lokasi bit citra pada index ke 8 sampai 9, diperoleh 11.

3. Menyusun semua nilai bit hasil pencocokan dari vektor. Pada kasus tersebut diperoleh kecocokan yaitu

   Vektor 11 14, menghasilkan 1011,

   Vektor 3 4, menghasilkan 01

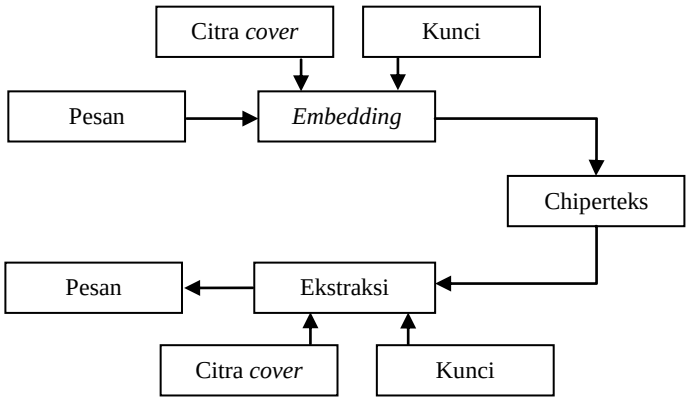
   Vektor 8 9, menghasilkan 11.

   Semua hasil diatas digabung menghasilkan *output* 10110111.

4. Jadi pesan yang diperoleh yaitu: 10110111.

3.5. Analisa Kombinasi Steganografi dan Kriptografi

Kombinasi steganografi dan kriptografi pada penelitian ini terdiri dari 2 proses, yaitu proses *embedding* dan ekstraksi yang secara umum dapat dilihat pada Gambar 3.5.

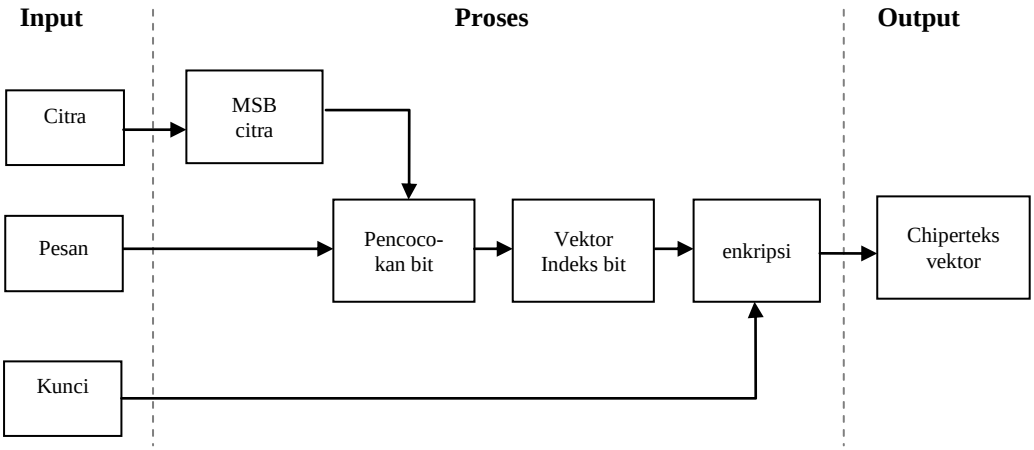


Gambar 0.5 Gambaran umum kombinasi steganografi dan kriptografi

Pada perancangan ini, peneliti mengusulkan metode steganografi yang dikembangkan Challita & Farhat (2011). Pesan tidak dienkrip terlebih dahulu, namun dilakukkan pencocokan antara bit pesan dengan bit citra. Bit citra yang digunakan adalah MSB citra. Hasil pencocokan berupa indeks bit yang kemudian dienkripsi. Untuk mengembalikan pesan, perlu melakukan dekripsi indeks bit dan dilanjutkan rekonstruksi pesan dengan citra yang sama. Proses *embedding* dan ekstraksi pada penelitian ini diuraikan sebagai berikut.

3.5.1. Bagan Proses *Embeeding*

Masukkan proses *embedding* berupa pesan, citra, dan kunci. Pada proses *embeeding* tahap yang dilakukan yaitu mencocokan bit pesan pada bit MSB citra. Hasil pencocokan disimpan dalam vektor yang memuat indeks lokasi bit. Tahap selanjutnya yaitu melakukan proses enkripsi pada vektor tersebut. *Output* yang dihasilkan adalah chiperteks vektor yang telah terenkripsi. Proses *embedding* ditunjukkan pada Gambar 3.5.



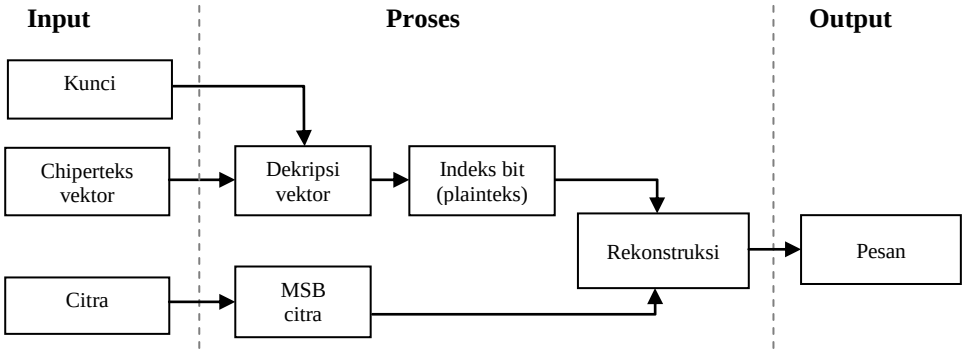
Gambar 0.6 Proses *embedding*

Langkah-langkah *embedding* adalah sebagai berikut:

a) Memasukkan *input* berupa citra, pesan, dan kunci.
b) Mengkonversi pesan dan citra dalam bentuk biner.
c) Mencocokan bit pesan dengan bit MSB citra. Posisi bit yang sama disimpan dalam vektor indeks bit.
d) Mengenkripsi vektor indeks bit dengan algoritma DES.
e) Hasil keluaran berupa chiperteks. Chiperteks tersebut memuat vektor indeks bit yang telah terenkripsi.
f) Selesai.

3.5.2. Bagan Proses Ekstraksi

Masukkan proses ekstraksi berupa chiperteks vektor, kunci, dan citra. Proses ekstraksi meliputi dekripsi vektor dan dilanjutkan rekonstruksi (lihat subbab 3.4.2). Hasil keluaran rekonstruksi berupa pesan semula. Bagan Proses ekstraksi ditunjukkan pada Gambar 3.7.



Gambar 0.7 Bagan proses ekstraksi

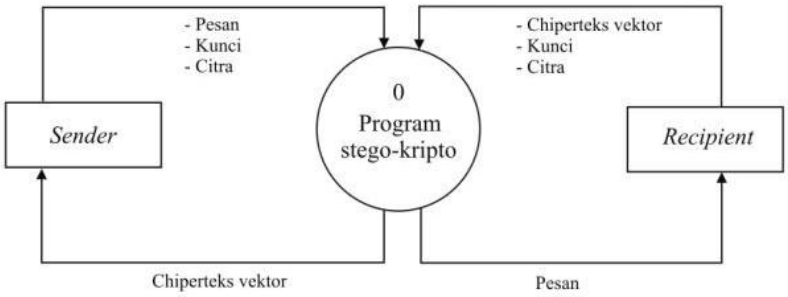
Langkah-langkah proses ekstraksi adalah sebagai berikut:

a) Memasukkan *input* berupa kunci, chiperteks vektor, dan citra.
b) Mendekripsi vektor dengan kunci, hasil dekripsi berupa plainteks indeks bit.
c) Melakukan rekonstruksi pesan dengan mencocokan bit MSB citra berdasar vektor indeks bit.
d) Hasil *output* berupa pesan.
e) Selesai.

3.6. Perancangan Aplikasi

3.6.1. Perancangan DFD (*Data Flow Diagram*)

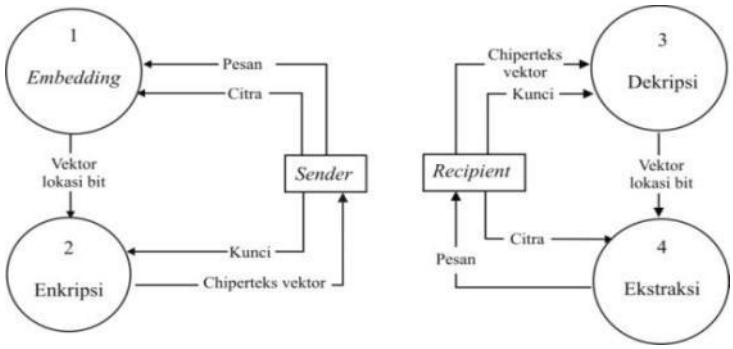
DFD atau diagram konteks kombinasi steganografi dan kriptografi pada penelitian ini terdiri dari DFD *level* 0 (Gambar 3.8) dan DFD *level* 1 (Gambar 3.9).



Gambar 0.8 Diagram konteks (DFD *level* 0)

Pada DFD *level* 0 terdapat dua entitas, yaitu *sender* (pengirim) dan *recipient* (penerima). *Sender* menginputkan pesan plainteks, kunci dan citra kemudian akan menerima keluaran berupa vektor lokasi bit yang telah dienkripsi (chiperteks). Sender mengirimkan chiperteks tersebut ke *recipient. Recipient* menginputkan chiperteks vektor, kunci dan citra. Hasil keluaran dari proses ini yaitu plainteks pesan yang dikirimkan oleh *sender*.

Proses tersebut selanjutnya diuraikan menjadi proses yang lebih rinci pada DFD level 1 yang diberikan pada Gambar 3.9.



Gambar 0.9 DFD *level* 1

Pada DFD *level* 1 terdiri dari 4 proses, yaitu *embedding*, enkripsi, dekripsi dan ekstraksi. Penjelasan masing-masing proses sebagai berikut:

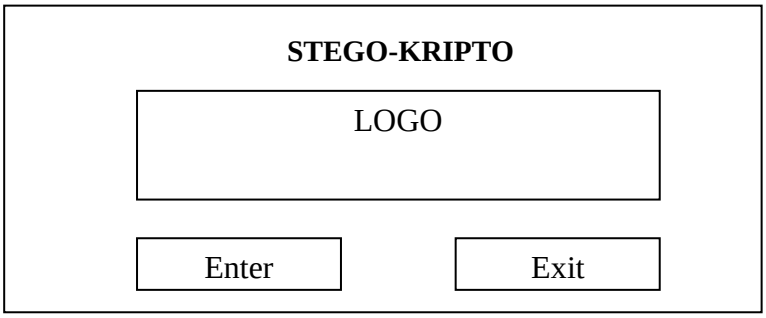
a. Proses 1 yaitu proses *embedding: sender* memberikan masukan berupa citra dan pesan. Pesan dikonversi kedalam bentuk biner dan dilakukan pencocokan susunan bit pesan pada citra. Hasil indeks lokasi bit yang sama disimpan dalam file vektor yang berisi indeks lokasi bit.
b. Proses 2 yaitu proses enkripsi: hasil vektor lokasi bit yang diperoleh pada proses 1 kemudian dienkripsi dengan algoritma DES. Keluaran pada proses ini yaitu vektor lokasi bit yang telah dienkripsi.
c. Proses 3 yaitu proses dekripsi: *recipient* memberikan masukkan berupa chiperteks vektor lokasi bit dan kunci. Keluaran pada proses ini yaitu plainteks vektor lokasi bit.
d. Proses 4 yaitu proses ekstraksi: yaitu proses pembentukan kembali pesan melalui pencocokan vektor lokasi bit dengan citra. Citra diinputkan oleh *recipient*, pada proses ini recipient memperoleh pesan asli yang dapat diketahui isinya.

3.6.2. Perancangan Antarmuka

Antarmuka merupakan halaman atau jendela (*form/window*) tempat *user* berinteraksi dengan aplikasi. Perancangan antarmuka bertujuan untuk memberikan gambaran desain dan tata letak menu yang akan digunakan. Perancangan antarmuka pada penelitian ini terdiri dari halaman awal, *embedding*, ekstraksi, dan *about*.

a. Halaman Awal

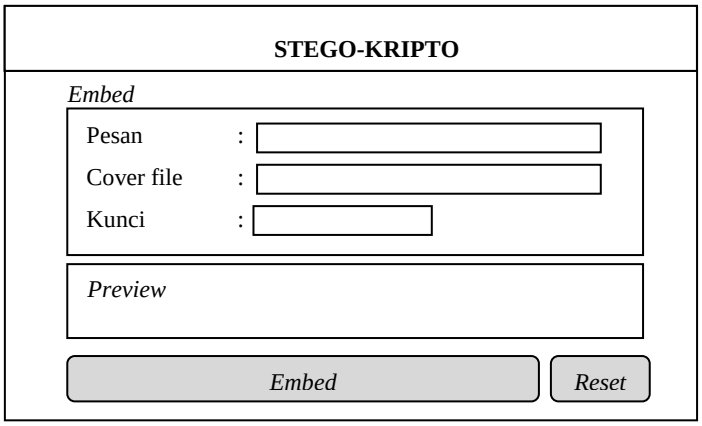
Halaman awal program terdiri dari logo dan 2 buah tombol menu, yaitu menu *Enter* yang digunakan untuk masuk ke program dan menu *Exit* untuk keluar dari program. Perancangan halaman awal ditunjukkan pada Gambar 3.10.



Gambar 0.10 Tampilan antramuka

b. *Embedding*

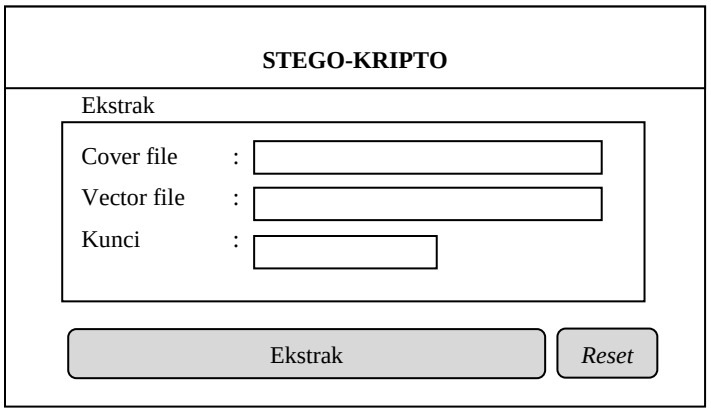
Pada *form embedding* terdapat 3 masukan, yaitu pesan, media *cover* dan kunci. Perancangan *form embedding* dapat dilihat pada Gambar 3.11.



Gambar 0.11 Perancangan *form embedding*

c. Ekstraksi

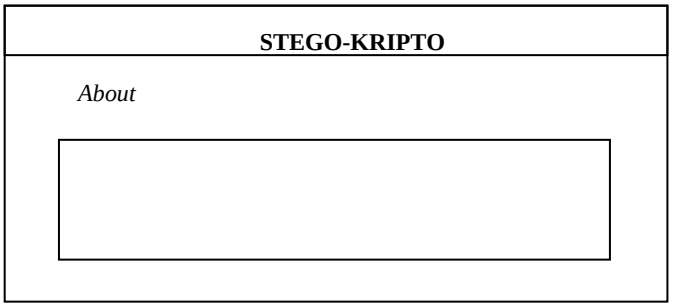
Pada *form* ekstraksi terdapat 3 masukan, yaitu pesan, media *cover* dan kunci. Perancangan *form* ekstraksi ditunjukkan pada Gambar 3.12.



Gambar 0.12 Perancangan *form* ekstraksi

d. *About*

Perancangan *form about* bertujuan untuk memberikan informasi seputar program. Desain *form about* ditunjukkan pada Gambar 3.13.



Gambar 0.13 Perancangan *form about*

3.7. Pengujian

Pengujian dilakukan dengan cara memberikan masukan (*input)* berupa pesan, citra, dan kunci. Kemudian dilakukan proses *embedding.* Keluaran *embedding* berupa chiperteks vektor yang memuat posisi indeks bit. Keluaran hasil *embedding* ini (vektor) selanjutnya menjadi masukan untuk proses ekstraksi. Proses ekstraksi diuji untuk memastikan pesan dapat dikembalikan seperti semula.

Adapun bahan pengujian yang digunakan diuraikan sebagai berikut.

1) Pesan teks

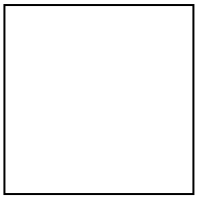
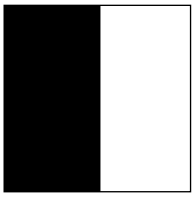
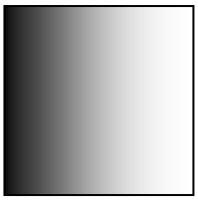
Teks pesan dapat berupa huruf, angka dan tanda baca seperti titik, koma, spasi dan karakter lainnya yang tersedia pada lampiran kode ASCII. Pesan teks yang digunakan berekstensi *.txt dengan ukuran 101 *bytes*, dengan isi pesan yaitu:

Transfer uang 50 juta via ATM.
Nomor rekening: 0123456
Password PIN: 9x8d7g
a.n. Budi Prasetiyo

2) Citra Hitam Putih

Karakteristik citra hitam-putih yang digunakan ditunjukkan pada Tabel 3.23. Pengujian awal dilakukan dengan resolusi 100px, kemudian dilakukan pengujian dengan berbagai ukuran mulai 512px,256px, 128px, dan 64px.

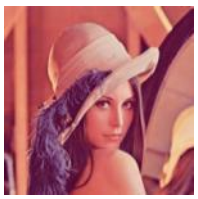
Tabel 0.23 Citra uji dengan kombinasai warna dasar hitam putih

No	Karakteristik citra	Citra	Ukuran
a.	Citra dengan komposisi warna 100 % putih		White.bmp 100px x 100px
b.	Citra dengan komposisi warna 100% hitam		Black.bmp 100px x 100px
c.	Citra dengan komposisi warna hitam putih blok		Block.bmp 100px x 100px
d.	Citra dengan warna hitam putih gradasi		Gradasi.bmp 100px x 100px

3) Citra Warna (*Color)*

Citra warna yang digunakan yaitu citra yang pada dasarnya sering digunakan sebagai bahan uji dalam berbagai penelitian pengolahan citra seperti ditunjukan pada Tabel 3.24. Citra tersebut memiliki warna dominan *Red* (R), *Green* (G), dan *Blue* (B). Pengujian awal dilakukan dengan resolusi 100px, kemudian dilakukan pengujian dengan berbagai ukuran mulai 512px,256px, 128px, dan 64px.

Tabel 0.24 Citra warna sebagai bahan uji

No	Karakteristik citra	Citra	Keterangan
a.	Citra dengan komposisi warna dominan *Red* (R)		Lenna.bmp

Lanjutan Tabel 3.24

b.	Citra dengan komposisi warna dominan *Green* (G)		Pepper.bmp
c.	Citra dengan warna komposisi dominan *Blue* (B)		Jet.bmp
d.	Citra yang memuat semua unsur *Red* (R), *Green* (G) dan *Blue* (B)		Baboon.bmp
e.	Citra foto		Foto.bmp

Sumber gambar: a,b,d (http://www.eecs.qmul.ac.uk/~phao/CIP/Images/)
c (http://www.hlevkin.com/TestImages/)

4) Pengujian citra dengan *Noise*

Citra yang digunakan pada proses *embedding* selanjutnya diberi *noise' salt and pepper'* dan *gaussian.* Citra diberi *noise* '*salt and pepper'* dengan standar deviasi *d*= 0,001; 0,005; 0,01; 0,05. Pemberian citra dengan *noise 'gaussian'* menggunakan *mean* nol dan standar deviasi *d*=0,001; 0,005; 0,01; 0,05. Citra yang telah diberi *noise* kemudian diuji pada proses ekstraksi untuk mengembalikan pesan. Pemberian *noise* bertujuan untuk mengetahui tingkat akurasi indeks bit pada saat rekonstruksi pesan dan mengetahui ketahanan citra terhadap perubahan.

Citra yang telah diberi noise kemudian diukur perubahan kualitasnya dengan MSE (*Mean Square Error*) PSNR (*Peak Signal to Noise Ratio*). dihitung yang dihitung dengan persamaan (3.1) dan (3.2).

$$MSE = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left| f(x,y) - g(x,y)\right|^2 \qquad (3.1)$$

$$PSNR = 20.\log_{10}\frac{R}{MSE} \qquad (3.2)$$

keterangan:

f(x,y) : Nilai intensitas citra asli pada koordinat (x,y)

g(x,y) : Nilai intensitas citra stego pada koordinat (x,y)

M : Banyaknya piksel vertikal pada citra

N : Banyaknya piksel horisontal pada citra

R : Nilai maksimal intensitas citra, R=255.

Notasi MSE menunjukkan nilai *Mean Square Error* dari citra. Semakin kecil nilai MSE semakin bagus. Semakin besar parameter PSNR berarti semakin mirip dengan citra asli. Citra dengan nilai PSNR >35 dB dapat dikatakan memiliki kualitas yang baik.