ANALISIS *AVALANCHE EFFECT* TERHADAP ALGORITMA KRIPTOGRAFI *DATA ENCRYPTION STANDARD* (DES) DAN *ADVANCE ENCRYPTION STANDARD* (AES) DALAM ENKRIPSI DATA

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

oleh:
TONI RIZA PAHLEVI
0310963038-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009

UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

ANALISIS *AVALANCHE EFFECT* TERHADAP ALGORITMA KRIPTOGRAFI *DATA ENCRYPTION STANDARD* (DES) DAN *ADVANCE ENCRYPTION STANDARD* (AES) DALAM ENKRIPSI DATA

Oleh:
TONI RIZA PAHLEVI
0310963038-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 3 Februari 2009
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Bayu Rahayudi, ST., MT	**Nanang Yudi Setiawan, ST**
NIP. 132 318 424	**NIP. 132 318 425**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya
Ketua,

Dr. Agus Suryanto, MSc
NIP. 132 126 049

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama **:** Toni Riza Pahlevi
NIM **:** 0310963038
Jurusan **:** Matematika
Penulis skripsi berjudul **:** Analisis *avalanche effect* terhadap algoritma kriptografi *data encryption standard* (DES) dan *advance encryption standard* (AES) dalam enkripsi data.

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 22 Desember 2008
Yang menyatakan,

(Toni Riza Pahlevi)
NIM. 0310963038

ANALISIS *AVALANCHE EFFECT* TERHADAP ALGORITMA KRIPTOGRAFI *DATA ENCRYPTION STANDARD* (DES) DAN *ADVANCE ENCRYPTION STANDARD* (AES) DALAM ENKRIPSI DATA

ABSTRAK

Pemilihan aplikasi yang berhubungan dengan keamanan informasi merupakan tanggung jawab yang penting untuk setiap sistem informasi elektronik, salah satu cara untuk menjaga keamanan informasi adalah dengan cara kriptografi. Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan dan integritas data. Dua jenis algoritma kriptografi antara lain *Data Encrytion Standard* (DES) dan *Advance Encryption Standard* (AES). Algoritma kriptografi ini menggunakan langkah-langkah matematika untuk merubah informasi menjadi *chiper* dan juga mengubah *chiper* tersebut kembali ke bentuk aslinya. Penelitian ini akan melakukan uji coba tentang besar nilai *Avalanche Effect* dari algoritma kriptografi DES dan AES. Setelah dilakukan uji coba secara berulang-ulang didapatkan hasil bahwa algoritma kriptografi AES memiliki nilai rata-rata *Avalanche Effect* yang lebih besar daripada algoritma kriptografi DES. Pada uji coba pertama AES mampu menghasilkan nilai rata-rata *Avalanche Effect* sebesar 50% sedangkan DES 22%. Pada uji coba kedua AES mampu menghasilkan nilai rata-rata *Avalanche Effect* sebesar 20% sedangkan DES 8%. Hal ini menunjukkan bahwa algoritma kriptografi AES memiliki nilai *Avalanche Effect* yang lebih baik daripada algoritma kriptografi DES.

AVALANCHE EFFECT ANALYSIS OF DATA ENCRYPTION STANDARD (DES) ALGORTHM AND ADVANCE ENCRYTION STANDARD (AES) ALGORITHM FOR DATA ENCRYPTION

ABSTRACT

The selective application related procedural safeguards is an important responsibility for every electronic information system, one way to secure the information is with cryptography. Data Encryption Standard (DES) and Advance Encryption Standard (AES) are two kinds of cryptography algorithm. The algorithms define the mathematical steps to transform information into a chiper and also transform the chiper back to original form. This research is aimed to analyze the avalanche effect value of DES and AES cryptography algorithm. Repeated tests show that the avalanche effect value of AES is bigger than DES. In the first test the avalanche effect value of AES up to 50% and 22% for DES. In the second test the avalanche effect value of AES up to 20% and 8% for DES. This Shows that AES cryptography algorithm has better avalanche effect value than DES cryptography algorithm.

Kata Pengantar

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahNya, penulis masih dapat belajar dan mengerjakan skripsi yang berjudul "Analisis *avalanche effect* terhadap algoritma kriptografi *data encryption standard* (DES) dan *advance encryption standard* (AES) dalam enkripsi data". Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

Dalam penyelesaian tugas akhir ini, penulis telah mendapat begitu banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Bayu Rahayudi, ST., MT sebagai Pembimbing I dan Nanang Yudi Setiawan, ST selaku pembimbing II. Terima kasih atas semua waktu dan bimbingan yang telah diberikan.
2. Segenap bapak dan ibu dosen yang telah mendidik dan mengamalkan ilmunya kepada penulis.
3. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya
4. Ayah, Ibu, dan Kakak. Terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
5. Crebow dengan si jago merahnya, Safitri, Aldo, Pepy, Ria, Ulan, Elly, Tyo', Dafi, Reza, Yudha, Rio, Boim terima kasih atas dukungan dan semangat yang tiada henti.
6. Sahabat- sahabat ilkomers `03 dan seluruh penghuni ilkom.
7. Pihak lain yang telah membantu terselesaikannnya skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Penulis sadari bahwa masih banyak kekurangan dalam laporan ini disebabkan oleh keterbatasan kemampuan dan pengalaman. Oleh karena itu Penulis sangat menghargai saran dan kritik yang sifatnya membangun demi perbaikan penulisan dan mutu isi skripsi ini untuk kelanjutan penelitian serupa di masa mendatang.

Penulis berharap semoga skripsi ini dapat memberikan manfaat kepada pembaca dan bisa diambil manfaatnya, baik oleh Penulis

selaku mahasiswa maupun pihak-pihak lain yang tertarik untuk menekuni pengembangan algoritma kriptografi.

Malang, 22 Desember 2008

Penulis

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

Bab I
PENDAHULUAN

1.1 Latar Belakang

Informasi telah menjadi suatu komoditi yang sangat penting. Pentingnya nilai dari sebuah informasi menyebabkan munculnya keinginan agar informasi tersebut hanya boleh diakses oleh pihak-pihak tertentu saja (Rahardjo, 1998). Oleh karena itu, masalah keamanan dan kerahasiaan informasi atau data menjadi aspek penting dari suatu sistem informasi.

Penyandian data merupakan salah satu cara yang dapat digunakan untuk mengamankan informasi. Penyandian data dapat dilakukan dengan kriptografi. Menurut Schneier (1996) kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan.

Plaintext adalah informasi yang dapat dibaca dan dimengerti maknanya dengan mudah. Enkripsi merupakan proses yang dilakukan untuk mengamankan *plaintext* menjadi informasi yang tersembunyi (*ciphertext*), dengan menggunakan kunci (*key*) tertentu. *Ciphertext* merupakan informasi yang tidak dapat dibaca dengan mudah. Agar *ciphertext* dapat dibaca sebagai *plaintext* maka dilakukan proses dekripsi (Schneier, 1996). Algoritma kriptografi (*cipher*) adalah persamaan matematika yang digunakan untuk melakukan enkripsi dan dekripsi (Munir, 2006).

Berdasarkan sejarah, kriptografi dibagi menjadi kriptografi klasik dan kriptografi modern. Kriptografi klasik merupakan kriptografi yang dilakukan sebelum adanya komputer. Sedangkan kriptografi modern merupakan kriptografi yang dilakukan setelah adanya komputer (Munir, 2006). Berdasar kunci yang digunakan untuk enkripsi dan dekripsi, kriptografi dibedakan lagi atas kriptografi kunci simetri dan kriptografi kunci asimetri. Kriptografi kunci simetri menggunakan kunci yang sama untuk proses enkripsi dan dekripsi, sedangkan kriptografi kunci asimetri sebaliknya (Kurniawan, 2004). Kriptografi kunci simetri disebut juga dengan kriptografi kunci privat, di mana keamanan sistemnya terletak pada kerahasiaan kuncinya. Semua algoritma kriptografi klasik termasuk dalam kriptografi kunci simetri. Selain algoritma kriptografi klasik

ada puluhan algoritma kriptografi modern yang termasuk dalam kriptografi simetri, di antaranya adalah DES (*Data Encryption Standard*) dan AES (*Advance Encryption Standard*).

Saat ini ada begitu banyak algoritma kriptografi yang telah dibuat dan dipublikasikan. Hal ini mengakibatkan munculnya kebingungan bagi pihak-pihak yang ingin mengamankan informasinya, dalam menentukan algoritma kriptografi mana yang lebih baik untuk diimplementasikan, terutama bagi pihak dengan sumber daya yang terbatas.

Beberapa parameter yang dapat digunakan untuk menentukan apakah suatu algoritma kriptografi dapat dikatakan lebih baik dari pada algoritma kriptografi yang lain adalah algoritma tersebut memiliki jumlah bit kunci yang cukup panjang, waktu yang diperlukan untuk melakukan enkripsi dan dekripsi cukup cepat, jumlah memori yang dibutuhkan selama proses enkripsi dan dekripsi sedikit, dan memiliki nilai *avalanche effect* yang besar. *Avalanche effect* merupakan nilai yang menunjukkan seberapa besar perubahan kecil yang dilakukan pada *plaintext* maupun *key* akan menyebabkan perubahan pada *ciphertext* yang dihasilkan (Budiyono, 2004). Semakin besar nilai *avalanche effect* maka semakin baik algoritma tersebut (Munir, 2006).

Berdasarkan latar belakang yang telah dikemukakan, maka dalam skripsi ini dilakukan analisis terhadap dua algoritma kriptografi kunci simetri yang cukup populer, yakni DES dan AES. Oleh karena itu, judul yang diambil dalam skripsi ini adalah ***"Analisis avalanche effect* terhadap algoritma kriptografi *Data Encryption Standard* (DES) dan *Advance Encryption Standard* (AES) dalam enkripsi data"**.

1.2 Rumusan Masalah

Rumusan masalah dalam skripsi ini adalah:

1. Bagaimana mengimplementasikan enkripsi dan dekripsi data dengan menggunakan algoritma kriptografi DES dan AES?
2. Bagaimana melakukan analisis terhadap algoritma kriptografi DES dan AES dengan menggunakan parameter nilai *avalanche effect*.nya?

1.3 Tujuan

Tujuan yang ingin dicapai dari pembuatan skripsi ini adalah:

1. Mengimplementasikan algoritma DES dan AES dalam perangkat lunak.
2. Mengukur dan membandingkan nilai *avalanche effect* dari proses enkripsi dengan menggunakan algoritma DES dan AES.
3. Mengukur dan membandingkan algoritma kriptografi mana yang memiliki nilai *avalanche effect* yang lebih baik setelah dilakukan serangkaian uji coba terhadap kedua algoritma tersebut.

1.4 Batasan Masalah

Batasan masalah dalam penulisan skripsi ini adalah:

1. Hanya membahas mengenai enkripsi algoritma DES dan AES.
2. Data yang digunakan berupa teks yang diinputkan pada perangkat lunak.
3. Parameter yang dianalisis terbatas pada nilai *avalanche effect.*

1.5 Manfaat

Manfaat yang diperoleh dari penulisan skripsi ini adalah memberikan perbandingan mengenai nilai *avalanche effect* dari masing-masing algoritma DES dan AES, dengan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pengerjaan skripsi ini yang nantinya bisa digunakan sebagai acuan untuk pengujian algoritma-algoritma enkripsi yang lain.

1.6 Metode Penyelesaian Masalah

Metode penyelesaian masalah yang dilakukan pada penelitian ini, yaitu :

1. Studi Literatur
   Membaca dan mempelajari beberapa literatur (jurnal, buku dan artikel dari *website*) mengenai kriptografi dan algoritma kriptografi kunci simetris DES dan AES.
2. Perancangan dan implementasi perangkat lunak
   Merancang dan membangun sebuah perangkat lunak yang mengimplementasikan proses enkripsi-dekripsi data dengan menggunakan algoritma kriptografi DES dan AES.

3. Uji coba dan analisis hasil implementasi
   Menganalisis hasil implementasi, yaitu nilai *avalanche effect* dari kedua algoritma kriptografi yang diuji, sehingga dapat ditarik kesimpulan berupa algoritma mana yang memiliki nilai *avalanche effect* yang lebih baik.

BAB II
TINJAUAN PUSTAKA

2.1 Aljabar Abstrak

Aljabar abstrak (*abstract algebra*) adalah cabang matematika yang mempelajari struktur aljabar seperti grup (*group*), cincin (*ring*), medan (*field*), ruang vektor, dan aljabar (www.wikipedia.org, 2006). Yang dimaksud dengan struktur aljabar adalah satu atau lebih himpunan dengan sejumlah operasi yang didefinisikan di dalamnya serta memenuhi beberapa aksioma. Dengan kata lain, aljabat abstrak mengkaji struktur aljabar dan sifat-sifatnya. Aljabar abstrak adalah dasar untuk teori pengkodean (*coding theory*) dan kriptografi (Munir, 2006).

2.1.1 *Group*

Group (G, *) terdiri dari himpunan G bersama-sama dengan operasi biner * pada G (G x G $\rightarrow$ G), yang memenuhi empat aksioma berikut:

(i) Tertutup: operasi biner * menghasilkan nilai di dalam G, yaitu untuk semua a dan b di dalam G, a * b juga berada di dalam G.
(ii) Asosiatif: untuk semua a, b, dan c di dalam G, (a * b) * c = a * (b * c).
(iii) Terdapat elemen identitas e sedemikian sehingga untuk semua a di dalam G, maka berlaku e * a = a * e = a.
(iv) Untuk semua a di dalam G, terdapat $a^{-1} \in$ G sedemikian sehingga a * a^{-1} = a^{-1} * a = e, yang dalam hal ini e adalah elemen identitas. a^{-1} disebut elemen inversi.

Operator * adalah istilah umum dan dapat menunjuk pada operasi penjumlahan, perkalian atau operasi matematika yang lain. Sebuah grup disebut abelian jika memenuhi kondisi tambahan berikut:
(v) Komutatif: a * b = b * a, untuk semua a dan b di dalam G.
(Stallings, 2005).
Contoh:

Penjumlahan dalam modulo 2 dengan G = {0, 1} dan operator biner adalah + adalah sebuah grup. Operasi biner dengan penjumlahan dalam modulo 2 didefinisikan sebagai Tabel *Cayley* yang dapat dilihat pada tabel 2.1.

Tabel 2.1 Tabel *Cayley*

+	0	1
0	0	1
1	1	2

(G, +) adalah sebuah grup karena memenuhi keempat aksioma tersebut, yakni:

(i) Tertutup: semua hasil operasi penjumlahan selalu menghasilkan nilai yang terdapat di dalam G (0 atau 1).

(ii) Asosiatif: operasi penjumlahan modulo 2 bersifat asosiatif yang berarti bahwa (a + b) + c = a + (b + c) untuk semua a, b, dan c elemen dari {0, 1}.

(iii) Elemen 0 adalah elemen identitas dan mempunyai sifat bahwa a + 0 = 0 + a = a.

(iv) Untuk semua elemen a di dalam G, elemen 0^{-1} adalah 0 dan elemen 1^{-1} adalah 1 sehingga 0 + 0 = 0 dan 1 + 1 = 0 (pada kebanyakan kasus, elemen inversi tidak selalu dirinya sendiri). Karena operatornya adalah +, maka elemen inversinya dinamakan juga elemen inverse penjumlahan (sering disimbolkan sebagai -a) (Munir, 2006).

2.1.2 *Ring*

Ring (R, +, x) terdiri dari himpunan R bersama-sama dengan dua operasi biner + dan x (masing-masing disebut penjumlahan dan perkalian) sedemikian sehingga memenuhi aksioma berikut:

(i) (R, +) adalah grup abelian dengan elemen identitas adalah 0.

(ii) Operasi perkalian bersifat asosiatif, yaitu a x (b x c) = (a x b) x c untuk semua a, b, c $\in$ R.

(iii) Terdapat elemen identitas perkalian yang dinyatakan dengan 1, dimana 1 $\neq$ 0, sedemikian sehingga 1 x a = a x 1 = a untuk semua a $\in$ R.

(iv) Operasi x bersifat distributif terhadap penjumlahan, yaitu:

a x (b + c) = (a x b) + (a x c) dan

(b + c) x a = (b x a) + (c x a)

untuk a, b, c $\in$ R (Stallings, 2005).

2.1.3 *Field*

Field F adalah sebuah *ring* komutatif dimana setiap elemen tidak nol mempunyai inversi perkalian. Yang dimaksud dengan inversi perkalian adalah untuk setiap a $\neq 0$ yang termasuk di dalam F, terdapat elemen $a^{-1} \in$ F sedemikian sehingga a x a^{-1} = 1. Secara sederhana *field* adalah tempat dimana dapat dilakukan operasi seperti penjumlahan dan perkalian (pengurangan a – b dipandang sebagai a + (-b) yang dalam hal ini –b adalah elemen inversi penjumlahan dari b, sedangkan a/b dipandang sebagai a x b^{-1} yang dalam hal ini b^{-1} adalah elemen inversi perkalian dari b).

Sebuah *field* disebut dengan *finite field* (medan berhingga) jika himpunannya memiliki jumlah elemen yang berhingga. Jumlah elemen di dalam *finite field* disebut dengan orde *field. Infinite field* (medan tak berhingga) adalah medan yang mempunyai jumlah elemen yang tak terbatas (Munir, 2006).

2.1.4 *Finite field* Fp

Untuk p bilangan prima, maka Fp adalah *finite field* (medan berhingga) berorde p dengan anggotanya adalah {0, 1, 2, …, p-1}, dimana operasi penjumlahan dan perkalian dilakukan dalam modulus p, yang didefinisikan sebagai berikut:

(i) Penjumlahan: jika a, b $\in$ Fp, maka a + b = r, yang dalam hal ini r = (a + b) mod p.
(ii) Perkalian: jika a, b $\in$ Fp, maka a x b = s, yang dalam hal ini s = (a x b) mod p.

Contoh:

Contoh *finite field* yag terkecil adalah dengan mengambil p = 2, yaitu F_2, yang hanya beranggotakan 0 dan 1. Contoh *finite field* lainnya adalah F_{23} yang mempunyai anggota {0, 1, 2, ..22}. Operasi penjumlahan dalam F_{23} misalnya 12 + 20 = 9 (karena 32 mod 23 = 9) dan operasi perkalian 8 x 9 = 3 (karena 72 mod 23 = 3).

Finite field mempunyai keunikan sebagai berikut:

(i) Jika F adalah *finite field*, maka F mengandung p^n elemen untuk beberapa bilangan prima p dan bilangan bulat n ≥ 1.
(ii) Untuk setiap bilangan prima p dan n ≥ 1 terdapat *finite field* dengan p^n elemen.

(iii) Untuk setiap perpangkatan bilangan prima p^n, n ≥ 1, terdapat *finite field* berorde p^n. *Field* ini sering dinyatakan sebagai GF(p^n). Gf adalah singkatan dari Galois *Field* (Munir, 2006).

2.1.5 *Galois field*

Finite field sering disebut juga dengan Galois *Field* (medan Galois), sebagai penghargaan terhadap Evariste Galois yang menemukan hubungan antara grup dan persamaan polinomial pada tahun 1832. Galois *field* adalah *finite field* dengan p^n elemen, yang dalam hal ini p adalah bilangan prima dan n adalah integer positif, $n \geq 1$ (Munir, 2006).

GF(p), dimana p adalah sebuah bilangan prima, merupakan *ring* sederhana dari integer modulo p. Untuk itu, dapat dilakukan operasi-operasi (penjumlahan, pengurangan, perkalian) menggunakan operasi yang lazim dalam integer, yang kemudian diikuti dengan reduksi modulo p. Sebagai contoh, dalam GF(5), 4 + 3 = 7 direduksi menjadi 2 mod 5. Sebuah kasus khusus adalah GF(2), dimana operasi penjumlahan adalah exclusive OR (XOR) dan operasi perkalian adalah AND (www.wikipedia.org, 2008).

Contoh:

GF(2)

+	0	1
0	0	1
1	1	0

x	0	1
0	0	0
1	0	1

GF(7)

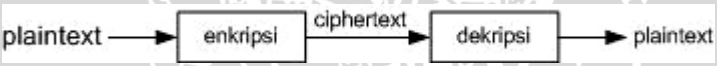
+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(Stallings, 2005).

2.2 Kriptografi

Menurut Schneier (1996), pesan (*message*) adalah data yang dapat dibaca dan dimengerti maknanya. Kata lain dari pesan adalah *plaintext.* Enkripsi merupakan proses untuk menyembunyikan pesan dalam bentuk lain yang tidak mudah dimengerti untuk menyembunyikan substansinya. Pesan yang telah dienkripsi disebut *ciphertext* dan proses untuk mengembalikan *ciphertext* menjadi *plaintext* disebut dekripsi. Keterkaitan tersebut digambarkan pada Gambar 2.1.



Gambar 2.1 Enkripsi dan dekripsi pada *plaintext* (Schneier, 1996)

Kriptografi (*cryptography*) adalah ilmu dan seni untuk menjaga keamanan pesan. Para praktisi kriptografi disebut kriptografer (*cryptographers*) (Schneier, 1996). Sebuah algoritma kriptografi disebut *cipher*, merupakan persamaan matematika yang digunakan untuk proses enkripsi dan dekripsi. Biasanya kedua persamaan matematika (untuk enkripsi dan dekripsi) memiliki hubungan matematis yang cukup erat (Rahardjo, 1998).

Konsep matematis yang mendasari algoritma kriptografi adalah relasi antara dua buah himpunan, yaitu himpunan yang berisi elemen-elemen *plaintext* dan himpunan yang berisi *ciphertext*. Misalkan P menyatakan *plaintext* dan C menyatakan *ciphertext*, maka fungsi enkripsi E memetakan P ke C,

$$E(P) = C \qquad (2.1)$$

dan fungsi dekripsi D memetakan C ke P,

$$D(C) = P \quad (2.2)$$

Karena proses enkripsi kemudian dekripsi mengembalikan pesan ke pesan awal, maka persamaan 2.3 harus benar:

$$D(E(P)) = P \quad (2.3)$$

(Munir, 2006).

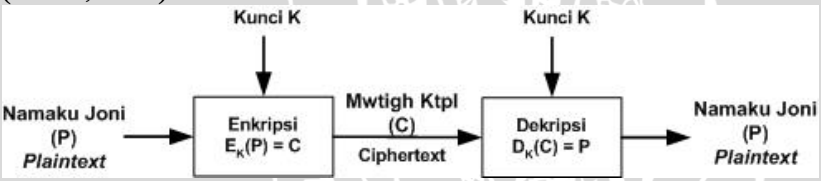
Pada kriptografi modern, algoritma kriptografi bersifat publik, tetapi kunci harus tetap dijaga kerahasiaannya. Kunci adalah parameter yang digunakan untuk transformasi enkripsi dan dekripsi. Kunci biasanya berupa *string* atau deretan bilangan. Dengan menggunakan kunci K, maka fungsi enkripsi dan dekripsi dapat ditulis seperti pada persamaan 2.4. Ilustrasinya dapat dilihat pada gambar 2.2.

$$E_K(P) = C \text{ dan } D_K(C) = P \quad (2.4)$$

dan kedua fungsi ini memenuhi:

$$D_K(E_K(P)) = P \quad (2.5)$$

(Munir, 2006).



Gambar 2.2 Ilustrasi enkripsi dan dekripsi pada pesan dengan menggunakan kunci K
(Munir, 2006)

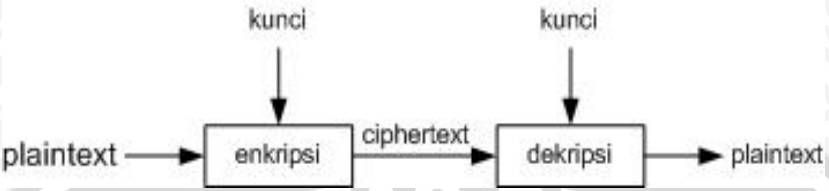
2.3 Algoritma Kriptografi

Algoritma kriptografi yang biasanya disebut dengan *cipher* adalah fungsi matematika yang digunakan untuk enkripsi dan dekripsi (Schneier, 1996). Berdasarkan kunci yang dipakai, algoritma kriptografi dapat dibedakan atas 2 jenis, yakni:

1. Algoritma kunci simetris (*symmetric-key cryptography*)

Dalam algoritma kunci simetris, kunci yang digunakan untuk melakukan enkripsi sama dengan kunci yang digunakan untuk dekripsi. Istilah lain yang digunakan untuk algoritma ini adalah algoritma kriptografi kunci privat. Keamanan kriptografi ini terletak pada kerahasiaan kuncinya. Contoh algoritma kunci simetris adalah DES (*Data Encryption Standard*), *Blowfish, Twofish,* dan *AES.*

Proses enkripsi dan dekripsi algoritma kunci simetris dapat dilihat pada gambar 2.3.

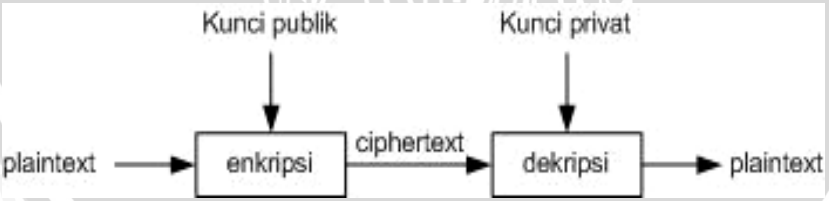


Gambar 2.3 Enkripsi dan dekripsi algoritma kunci simetris (Schneier, 1996)

Algoritma kriptografi simetris dibagi menjadi dua yaitu algoritma aliran (*stream ciphers*) dan algoritma blok (*block ciphers*). Pada algoritma aliran, proses enkripsi berorientasi pada satu bit atau satu byte data. Sedangkan untuk algoritma blok, proses enkripsi berorientasi pada sekumpulan bit atau *byte* data (per blok).

2. Algoritma kunci asimetris

Jika kunci yang digunakan untuk enkripsi tidak sama dengan kunci untuk dekripsi, maka algoritma kriptografinya disebut algoritma kunci asimetris. Algoritma ini disebut juga algoritma kriptografi kunci publik. Semua orang yang mendapatkan kunci publik dapat menggunakannya untuk mengenkripsikan suatu pesan, data ataupun informasi, namun hanya satu orang saja yang memiliki kunci privat untuk mendekripsikan *ciphertext* yang diterimanya menjadi *plaintext*. Contoh dari algoritma ini adalah RSA, *ElGamal* dan *Knapsack*. Proses enkripsi dan dekripsi algoritma kunci asimetris dapat dilihat pada gambar 2.4.



Gambar 2.4 Enkripsi dan dekripsi algoritma kunci asimetris (Munir, 2006)

2.4 DES

2.4.1 Struktur DES

DES merupakan nama dari sebuah algoritma untuk mengenkripsi data yang dikeluarkan oleh *Federal Information Processing Standard* (FIPS) 46 – 1 Amerika Serikat (Nugraha, 1999). DES termasuk algoritma *block cipher* yang dijadikan standar enkripsi kunci simetris pertama.

DES beroperasi pada ukuran blok 64 bit. DES mengenkripsikan 64 bit *plaintext* menjadi 64 bit *ciphertext* dengan menggunakan 56 bit kunci internal yang dibangkitkan dari 64 bit kunci eksternal (Munir, 2006). Kunci eksternal merupakan kunci yang dimasukkan oleh *user* pada sistem. Kunci internal merupakan kunci yang digunakan untuk melakukan enkripsi pada setiap putaran DES, yang diperoleh dari kunci eksternal yang telah diproses.

2.4.2 Proses inisialisasi kunci internal

DES menggunakan 56 bit kunci untuk melakukan enkripsi dan dekripsi blok masukan. Kunci internal dibangkitkan dari kunci eksternal yang diberikan oleh *user*. Kunci eksternal tersebut panjangnya 64 bit (8 *byte*) atau 8 karakter. Namun tiap bit ke-8 (*parity bits*) dari 8 *byte* kunci diabaikan. Oleh karena itu, hanya ada 56 bit kunci yang digunakan (*effective bits*). Permutasi yang dilakukan pada bit-bit ini disebut dengan *permuted choice* 1 (PC-1), tabel *permuted choice* 1 (PC-1) dapat dilihat pada tabel 2.2. *Permuted choice* 1 ini mengubah posisi bit-bit pada kunci eksternal dan mengurangi bit kunci eksternal, yang awalnya berjumlah 64 bit menjadi 56 bit dengan cara menghilangkan *parity bits* (Bakhtiari, 1994). Pada tabel 2.2 setiap elemen menunjukkan posisi bit masukan untuk PC-1 yang sesuai atau berkoresponden dengan posisi bit keluaran (i+j). Misalnya, bit ke-60 dari bit kunci eksternal merupakan bit ke-0 dari bit kunci internal.

Kunci internal yang diperoleh dibagi menjadi 2 bagian, kiri (C) dan kanan (D), yang masing-masing panjangnya 28 bit. D terdiri dari bit pada posisi ke-0 hingga ke-27, sedangkan C terdiri dari bit pada posisi ke-28 hingga ke-55. Kedua bagian digeser ke kiri (*left shift)* sepanjang 1 atau 2 bit bergantung pada tiap putaran, tabel pergeseran bit pada setiap putaran dapat dilihat pada tabel 2.3. Setelah pergeseran bit, dilakukan permutasi (*permuted choice* 2) pada hasil

pergeseran bit dari kedua bagian tersebut untuk memperoleh kunci internal pada tiap putaran, yang selanjutnya disebut dengan *subkey*.

Tabel 2.2 *Permuted choice 1* (PC-1) (Bakhtiari, 1994)

i\j	0	1	2	3	4	5	6
0	60	52	44	36	59	51	43
7	35	27	19	11	3	58	50
14	42	34	26	18	10	2	27
21	49	41	33	25	17	9	1
28	28	20	12	4	61	53	45
35	37	29	21	13	5	62	54
42	46	38	30	22	14	6	63
49	55	47	39	31	23	15	7

Tabel 2.3 Jumlah pergeseran bit setiap putaran (Bakhtiari, 1994)

Putaran ke-	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

Tabel 2.4 *Permuted choice 2* (PC-2) (Bakhtiari, 1994)

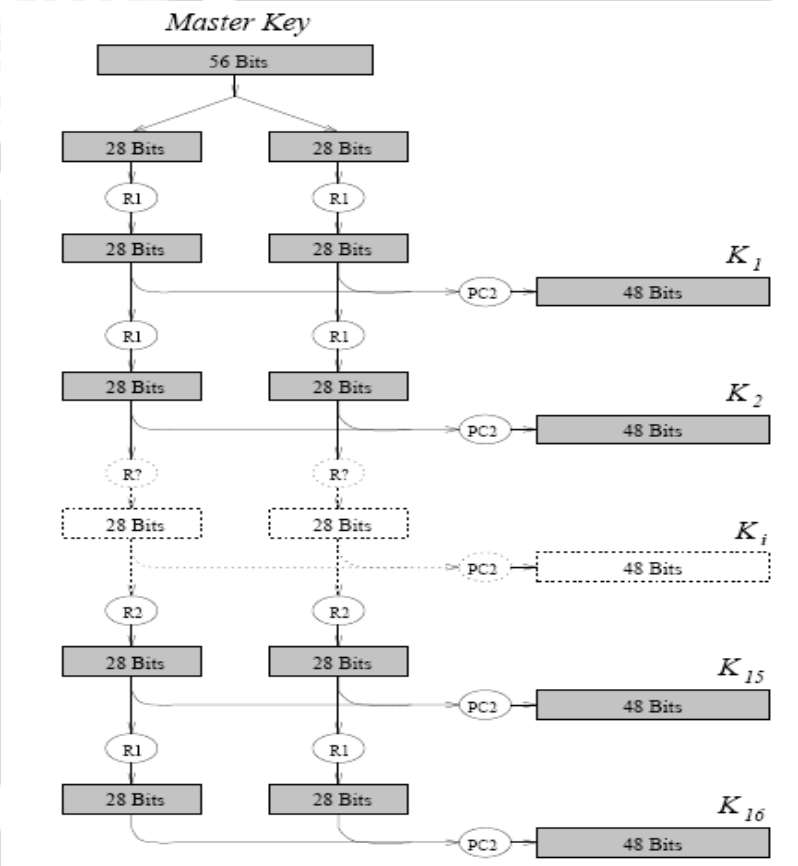
$i\backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	24	27	20	6	14	10	3	22	0	17	7	12	8	23	11	5
16	16	26	1	9	19	25	4	15	54	43	36	29	49	40	48	30
32	52	44	37	33	46	35	50	41	28	53	51	55	32	45	39	42

Setiap elemen pada tabel *permuted choice 2* pada tabel 2.4 menunjukkan posisi bit masukan yang sesuai atau berkoresponden dengan posisi bit keluaran (i+j).
Contoh proses inisialisasi kunci.

- Kunci eksternal = PASSWORD, dengan
  P = 01010000
  A = 01000001
  S = 01010011
  S = 01010011
  W = 01010111
  O = 01001111
  R = 01010010
  D = 01000100
  Jika dituliskan dalam bentuk biner, PASSWORD menjadi:
  01010000 01000001 01010011 01010011 01010111 01001111 01010010 01000100
- Kemudian dilakukan *permuted choice* 1 pada kunci eksternal untuk memperoleh 56 bit kunci internal. Dengan menggunakan tabel 2.1, bit ke-0 (bit yang berada paling kanan dari keseluruhan blok bit) dari 56 bit kunci internal merupakan bit ke-60 pada kunci eksternal. Bit ke-1 dari bit kunci internal merupakan bit ke-52 pada kunci eksternal, demikian seterusnya. Hasilnya menjadi:
  0000000 0111111 1100000 0000101 0111110 0101100 0000100 0001101
- Blok bit ini dibagi menjadi 2 bagian, yang terdiri atas
  C0 = 0000000 0111111 1100000 0000101
  D0 = 0111110 0101100 0000100 0001101
  Dari C0 dan D0 dilakukan pergeseran bit yang jumlahnya sesuai dengan tabel 2.2. C0 dan D0 menjadi
  C1 = 0000000 1111111 1000000 0001010
  D1 = 1111100 1011000 0001000 0011010
- Untuk memperoleh K_1, dilakukan *permuted choice* 2 pada C1 dan D1, diperoleh:
  K_1 = 101000 001001 001001 001010 011001 100010 001001 100111
- K_2 diperoleh dari *permuted choice* 2 yang dilakukan pada C1 dan D1 yang digeser 1 bit ke kiri.

C2 = 0000001 1111111 0000000 0010100
D2 = 1111001 0110000 0010000 0110101
K_2 = 101100 000001 001011 010010 011100 101100 000100 000011

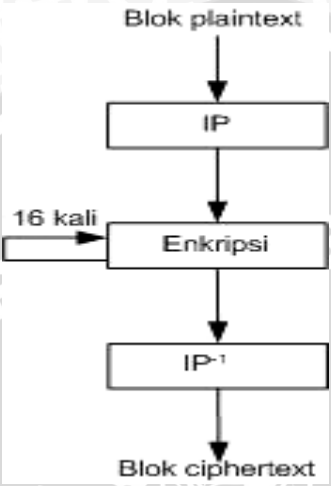
- Begitu seterusnya hingga K_{16}.



Gambar 2.5 Proses penjadwalan kunci
(Bakhtiari, 1994)

R1 menandakan pergeseran 1 bit ke kiri
R2 menandakan pergeseran 2 bit ke kiri
PC2 menandakan *permutation choice* 2

2.4.3 Enkripsi

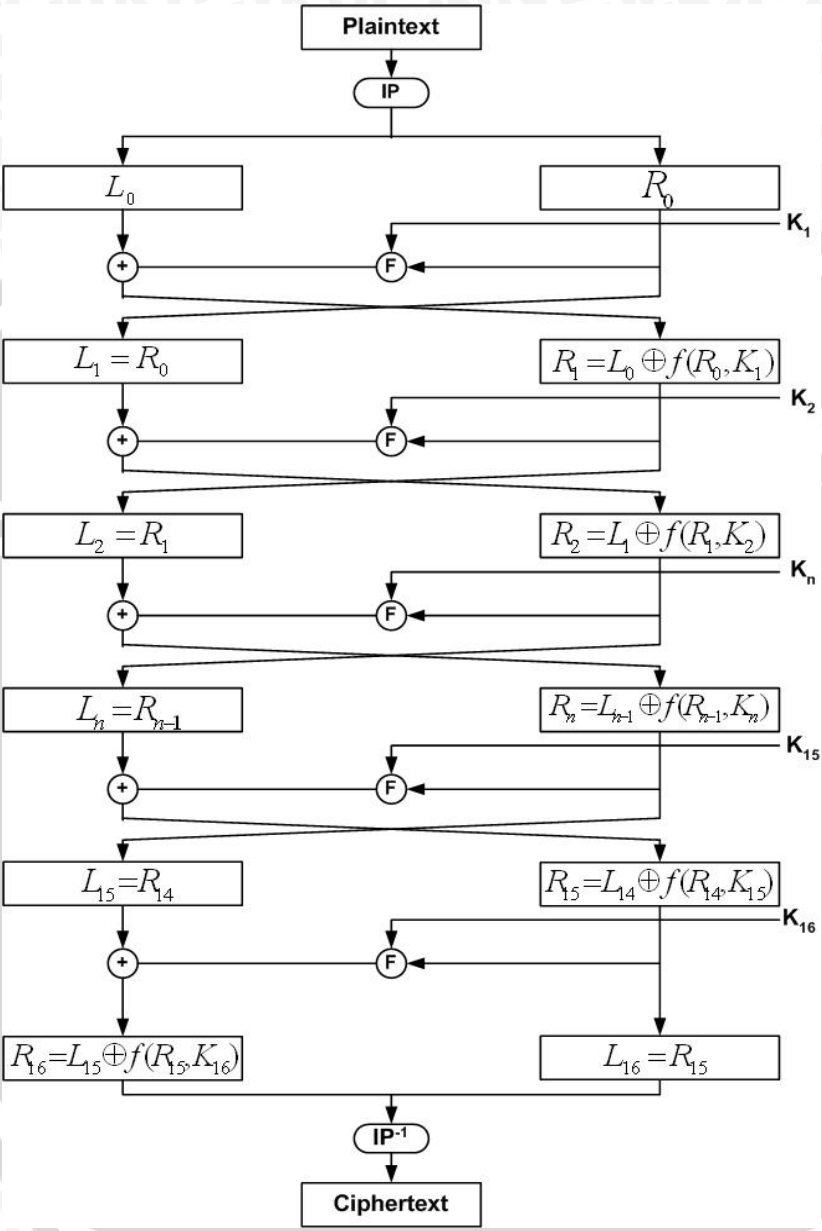


Gambar 2.6 Skema global algortima DES
(Munir, 2006)

Dari Gambar 2.6, dapat dituliskan skema global dari algoritma DES sebagai berikut:

1. Blok *plaintext* dipermutasi dengan matrik permutasi awal (*initial permutation* atau IP). *Plaintext* merupakan data (teks) yang akan dienkripsi. *Plaintext* ini direpresentasikan sebagai bit, misalnya huruf P direpresentasikan sebagai 01010000. Keseluruhan *plaintext* dibagi ke dalam blok-blok. Setiap blok terdiri atas 64 bit.
2. Hasil permutasi awal kemudian dienkripsikan sebanyak 16 kali (16 putaran). Setiap putaran menggunakan kunci internal yang berbeda.
3. Hasil enkripsi kemudian dipermutasikan dengan matriks balikan (*invers initial permutation* atau IP^{-1}) menjadi blok *ciphertext* (Munir, 2006).

Gambar 2.7 memperlihatkan proses enkripsi algoritma DES. Algoritma DES menggunakan *initial permutation* (IP) pada awal dan invers dari *initial permutation* (IP^{-1}) di akhir algoritma. Permutasi ini hanya mengubah posisi bit-bit pada blok masukan dan keluaran dari algoritma DES.



Gambar 2.7 Proses enkripsi algoritma DES
(FIPS PUB, 1999)

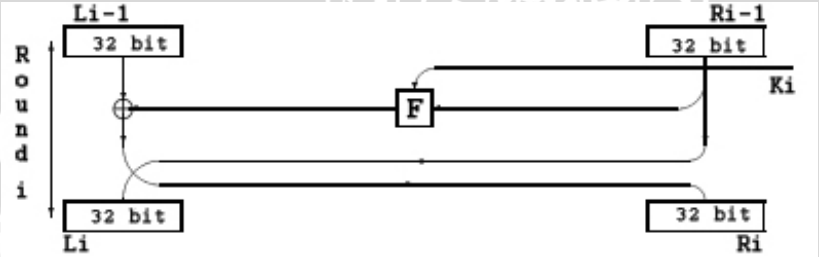
Sebelum proses enkripsi dilakukan, akan dilakukan permutasi awal (*Initial Permutation*) terhadap *plaintext*. Tujuan dari permutasi awal adalah mengacak *plaintext* sehingga urutan bit-bit di dalamnya berubah. Pengacakan dilakukan dengan menggunakan matriks permutasi awal pada Tabel 2.5 (Munir, 2006).

Tabel 2.5 *Initial Permutation* (Permutasi Awal) (FIPS PUB, 1999)

i\j	0	1	2	3	4	5	6	7	8	9
0	57	49	41	33	25	17	9	1	59	51
10	43	35	27	19	11	3	61	53	45	37
20	29	21	13	5	63	55	47	39	31	23
30	15	7	56	48	40	32	24	16	8	0
40	58	50	42	34	26	18	10	2	60	52
50	44	36	28	20	12	4	62	54	46	38
60	30	22	14	6						

Setiap elemen pada Tabel *Initial Permutation* (Permutasi Awal) menunjukkan posisi bit masukan yang sesuai atau berkoresponden dengan posisi bit keluaran (i+j). Artinya, bit ke-0 data hasil diisi dengan bit ke-57 data awal, bit ke-1 data hasil diisi dengan bit ke-49 data awal, demikian seterusnya (Kurniawan, 2006).

Setiap putaran DES menggunakan fungsi F. Gambar 2.8 menunjukkan satu putaran algoritma DES.



Gambar 2.8 Satu putaran DES
(Bakhtiari, 1994)

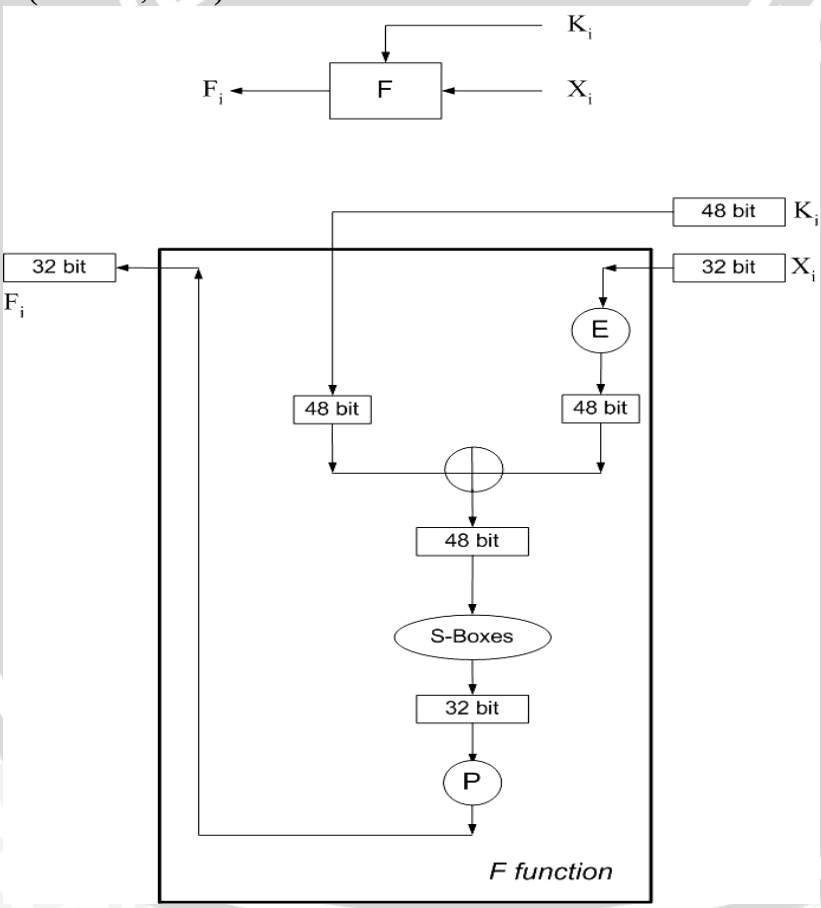
Dari gambar 2.8 dapat dijelaskan bahwa dalam proses enkripsi, blok *plaintext* terbagi menjadi dua bagian, kiri (L) dan kanan (R), yang masing-masing panjangnya 32 bit. Kedua bagian ini masuk ke dalam

16 putaran DES. Pada setiap putaran ke-i, blok R merupakan masukan untuk fungsi F, skema fungsi F dapat dilihat pada gambar 2.9. Pada fungsi F, blok R dikombinasikan dengan kunci internal K_i (kunci internal pada putaran ke-i). Keluaran dari fungsi F di XOR kan dengan blok L untuk mendapatkan blok R yang baru. Sedangkan blok L yang baru, langsung diambil dari blok R sebelumnya. Secara matematis, satu putaran DES dinyatakan dengan persamaan 2.6 dan 2.7.

$$L_i = R_{i-1} \quad (2.6)$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \quad (2.7)$$

(Schneier, 1996).



Gambar 2.9 Fungsi F (Bakhtiari, 1994)

X_i merupakan blok masukan untuk fungsi F pada putaran ke-i. E adalah fungsi ekspansi yang memperluas blok R_{i-1} yang panjangnya 32 bit menjadi blok 48 bit dengan menggunakan *Expansion function E* pada tabel 2.6.

Tabel 2.6 *Expansion function E* (Bakhtiari, 1994)

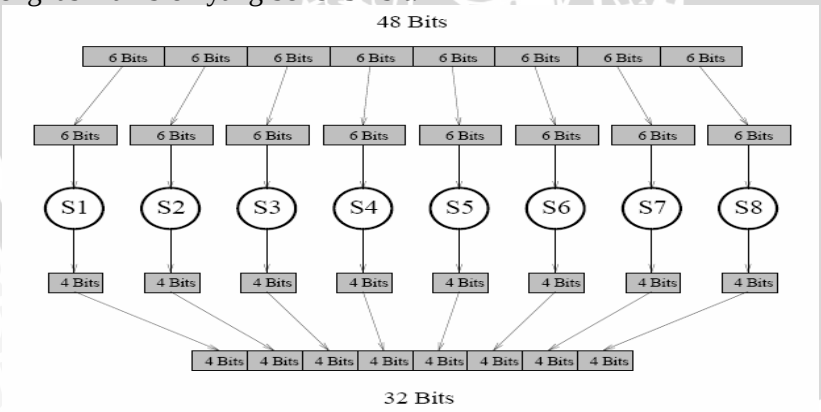
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	31	0	1	2	3	4	3	4	5	6	7	8	7	8	9	10
16	11	12	11	12	13	14	15	16	15	16	17	18	19	20	19	20
32	21	22	23	24	23	24	25	26	27	28	27	28	29	30	31	0

Sama halnya dengan tabel *permuted choice* 2, masing-masing elemen pada tabel *expansion function E* menunjukkan posisi bit masukan yang sesuai atau berkoresponden dengan posisi bit keluaran (i+j). Hasil ekspansi, yaitu $E(R_{i-1})$, yang panjangnya 48 bit di XOR kan dengan K_i yang panjangnya 48 bit dan menghasilkan *A* yang panjangnya juga 48 bit, atau dapat ditulis seperti pada persamaan 2.8.

$$E(R_{i-1}) \oplus K_i = A \qquad (2.8)$$

(Munir, 2006).

A, yang merupakan masukan bagi *S-Box*, dibagi menjadi 8 blok dimana setiap blok terdiri atas 6 bit pada gambar 2.10. Terdapat 8 buah *S-Box* dan setiap *S-Box* menerima masukan 6 bit dan menghasilkan keluaran 4 bit. Rangkaian keluaran dari setiap *S-Box* menghasilkan blok yang berisi 32 bit.



Gambar 2.10 Gambaran masukan dan keluaran *S-Box* (Bakhtiari, 1994)

Pada tabel *S-Box*, setiap elemen mengacu pada nilai yang diwakili oleh 4 bit keluaran, dan i+j mengacu pada nilai yang diwakili oleh 6 bit masukan *S-Box*. Tabel 2.7 merupakan 8 tabel *S-Box*.

Tabel 2.7 *S-Box* (Bakhtiari, 1994)

S-Box Number One																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	0	4	15	13	7	1	4	2	14	15	2	11	13	8	1
16	3	10	10	6	6	12	12	11	5	9	9	5	0	3	7	8
32	4	15	1	12	14	8	8	2	13	4	6	9	2	1	11	7
48	15	5	12	11	9	3	7	14	3	10	10	0	5	6	0	13

S-Box Number Two																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	3	1	13	8	4	14	7	6	15	11	2	3	8	4	14
16	9	12	7	0	2	1	13	10	12	6	0	9	5	11	10	5
32	0	13	14	8	7	10	11	1	10	3	4	15	13	4	1	2
48	5	11	8	6	12	7	6	12	9	0	3	5	2	14	15	9

S-Box Number Three																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	13	0	7	9	0	14	9	6	3	3	4	15	6	5	10
16	1	2	13	8	12	5	7	14	11	12	4	11	2	15	8	1
32	13	1	6	10	4	13	9	0	8	6	15	9	3	8	0	7
48	11	4	1	15	2	14	12	3	5	11	10	5	14	2	7	12

S-Box Number Four																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	13	8	14	11	3	5	0	6	6	15	9	0	10	3
16	1	4	2	7	8	2	5	12	11	1	12	10	4	14	15	9
32	10	3	6	15	9	0	0	6	12	10	11	1	7	13	13	8
48	15	9	1	4	3	5	14	11	5	12	2	7	8	2	4	14

S-Box Number Five																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	14	12	11	4	2	1	12	7	4	10	7	11	13	6	1
16	8	5	5	0	3	15	15	10	13	3	0	9	14	8	9	6
32	4	11	2	8	1	12	11	7	10	1	13	14	7	2	8	13
48	15	6	9	15	12	0	5	9	6	10	3	4	0	5	14	3

S-Box Number Six																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	10	1	15	10	4	15	2	9	7	2	12	6	9	8	5
16	0	6	13	1	3	13	4	14	14	0	7	11	5	3	13	8
32	9	4	14	3	15	2	5	12	2	9	8	5	12	15	3	10
48	7	11	0	14	4	1	10	7	1	6	13	0	11	8	6	13

S-Box Number Seven																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	13	11	0	2	11	14	7	15	4	0	9	8	1	13	10
16	3	14	12	3	9	5	7	12	5	2	10	15	6	8	1	6
32	1	6	4	11	11	13	13	8	12	1	3	4	7	10	14	7
48	10	9	15	5	6	0	8	15	0	14	5	2	9	3	2	12

S-Box Number Eight																
i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	1	2	15	8	13	4	8	6	10	15	3	11	7	1	4
16	10	12	9	5	3	6	14	11	5	0	0	14	12	9	7	2
32	7	2	11	1	4	14	1	7	9	4	12	10	14	8	2	13
48	0	15	6	12	10	9	13	0	15	3	3	5	5	6	8	11

Keluaran dari 8 *S-Box* yang panjangnya 32 bit menjadi masukan untuk proses permutasi. Tujuan dari permutasi adalah untuk mengacak bit-bit hasil *S-Box* dengan menggunakan matriks permutasi P (*P-Box*) tabel 2.8.

Tabel 2.8 Fungsi permutasi P (Bakhtiari, 1994)

i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	28	21	10	26	2	19	13	23	29	5	0	18	8	24	30
16	22	1	14	27	6	9	17	31	15	4	20	3	11	12	25	16

Setelah proses 16 iterasi proses enkripsi dilakukan, hasil dari bagian kanan dan kiri digabungkan kembali menjadi rangkaian bit sebesar 64 bit. Kemudian, hasil tersebut dipermutasi dengan Permutasi Awal Balikan (*Invers Initial Permutation* IP^{-1}) pada Tabel 2.9.

Tabel 2.9 *Invers Initial Permutation* IP^{-1}(FIPS PUB, 1999)

i\j	0	1	2	3	4	5	6	7	8	9
0	39	7	47	15	55	23	63	31	38	6
10	46	14	54	22	62	30	37	5	45	13
20	53	21	61	29	36	4	44	12	52	20
30	60	28	35	3	43	11	51	19	59	27
40	34	2	42	10	50	18	58	26	33	1
50	41	9	49	17	57	25	32	0	40	8
60	48	16	56	24						

Setiap elemen pada Tabel *Invers Initial Permutation* menunjukkan posisi bit masukan yang sesuai atau berkoresponden dengan posisi bit keluaran (i+j). Artinya, bit ke-0 data hasil diisi dengan bit ke-39 data awal, bit ke-1 data hasil diisi dengan bit ke-7 data awal, demikian seterusnya (Kurniawan, 2006). Hasil dari proses *Invers Initial Permutation* ini langsung menjadi *ciphertext* dengan ukuran sebesar 64 bit.

2.4.4 Dekripsi

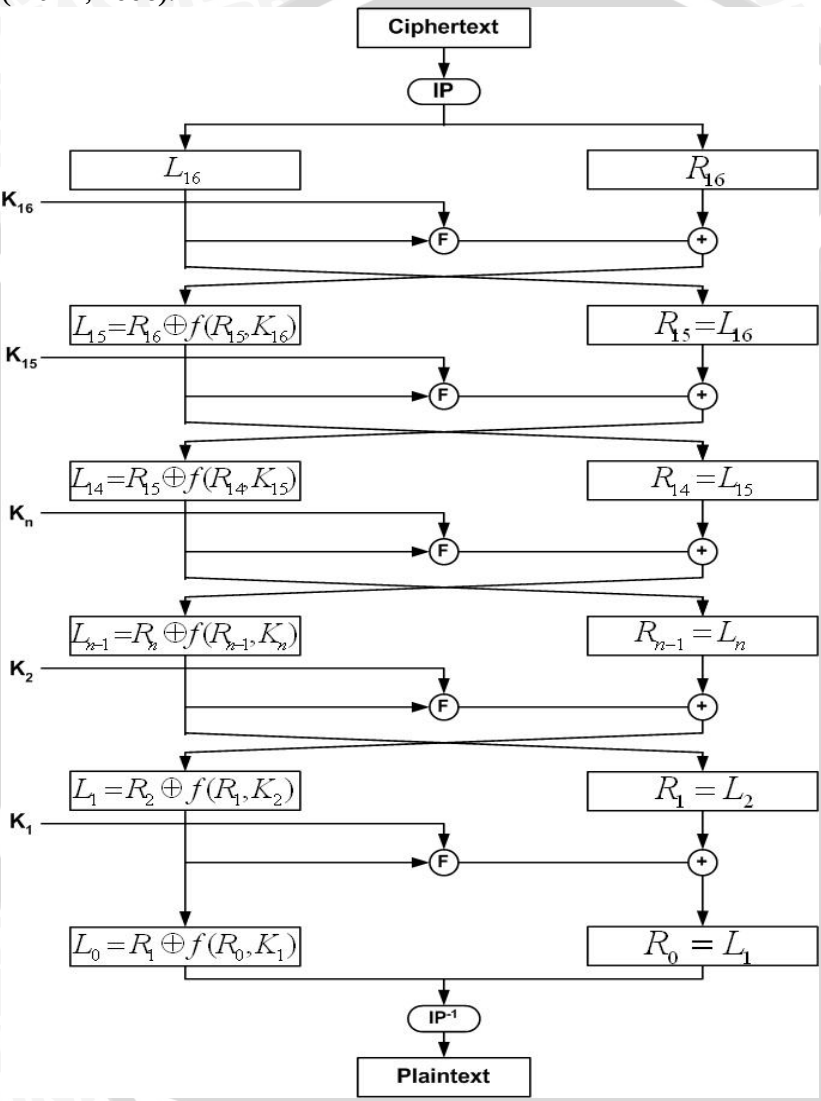
Proses dekripsi terhadap *ciphertext* merupakan kebalikan dari proses enkripsi gambar 2.11. *Data Encryption Standard* menggunakan algoritma yang sama untuk proses enkripsi dan dekripsi. Jika pada proses enkripsi urutan kunci internal yang digunakan adalah K_1, K_2, …., K_{16}, maka pada proses dekripsi urutan kunci yang digunakan adalah K_{16}, K_{15}, …., K_1. Untuk setiap putaran 16, 15, …, 1, keluaran pada setiap proses dekripsi adalah

$$L_i = R_{i-1} \qquad (2.9)$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \qquad (2.10)$$

Yang dalam hal ini (R_{16}, L_{16}) adalah blok masukan awal untuk dekripsi. Blok (R_{16}, L_{16}) diperoleh dengan mempermutasikan *ciphertext* dengan matriks permutasi IP^{-1}. Pergeseran bit ke kiri (*left shift*) yang dilakukan pada proses inisialisasi kunci internal, pada

proses dekripsi diganti menjadi pergeseran bit ke kanan (*right shift*). Pra keluaran dari proses dekripsi adalah (L_0 dan R_0). Dengan permutasi awal IP akan didapatkan kembali blok *plaintext* semula (Munir, 2006).



Gambar 2.11 Proses dekripsi algoritma DES
(Kurniawan, 2007)

2.5 AES

2.5.1 Input dan output

Input dan output untuk AES masing-masing terdiri dari deret 128 bit (digit-digit dengan nilai 0 atau 1). Deret ini kadang disebut sebagai blok dan jumlah bit di dalamnya disebut sebagai panjangnya. *Cipher key* (*key* yang digunakan untuk enkripsi atau dekripsi) untuk AES adalah deret 128, 192 atau 256 bit. Panjang input, output atau *key* yang berbeda tidak diizinkan dalam standar ini.

Bit-bit di dalam deret ini dihitung mulai 0 dan berahir pada panjang deret dikurangi 1. Angka i yang diberikan pada sebuah bit disebut juga sebagai indeksnya, dan akan berada dalam *range* $0 \le i < 128$, $0 \le i < 192$, atau $0 \le i < 256$ tergantung dari panjang blok atau kunci (FIPS 197, 2001).

2.5.2 *Bytes*

Unit dasar untuk pemrosesan dalam AES adalah byte, sebuah deret yang terdiri dari 8 bit yang dianggap sebagai satu entitas. Deret input, output dan *key* diproses sebagai *array bytes* yang dibentuk dengan cara membagi deret tadi menjadi grup-grup 8 bit berurutan untuk membentuk *array bytes* (lihat 2.5.3). Untuk sebuah input, output atau *key* yang disimbolkan dengan a, bytes dalam *array* yang dihasilkan ditunjuk menggunakan dua jenis bentuk, a_n atau a[n], dimana n ada di antara *range* berikut:

panjang *key* = 128 bit, $0 \le n < 16;$

panjang *key* = 192 bit, $0 \le n < 24;$

panjang *key* = 256 bit, $0 \le n < 32;$

panjang blok = 128 bit, $0 \le n < 16.$

Semua nilai byte dalam AES ditampilkan sebagai gabungan tiap-tiap bit individual (0 atau 1) di dalam tanda kurung dengan urutan $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Byte-byte ini diterjemahkan sebagai elemen *finite field* menggunakan representasi polinomial:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0 = \sum_{i=0}^{7} b_i x^i$$

Misalnya {01100011} mengidentifikasi elemen *finite field* yang spesifik yaitu $x^6 + x^5 + x + 1$ (FIPS 197, 2001).

Cukup mudah juga untuk menampilkan nilai byte menggunakan notasi heksadesimal dengan tiap-tiap dua grup 4 bit direpresentasikan oleh satu karakter, dapat dilihat pada tabel 2.10.

Tabel 2.10 Representasi heksadesimal dari susunan bit

Susunan bit	heksadesimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Maka berdasarkan Tabel 2.10, elemen {01100011} dapat direpresentasikan sebagai {63}, dimana karakter yang menunjukkan grup 4 bit dengan bit bernilai tinggi berada di kiri. Beberapa operasi *finite field* melibatkan satu bit tambahan (b_8) di ujung kiri dari sebuah byte yang tersusun dari 8 bit. Saat bit tambahan ini ada, ia akan tampil sebagai {01} sebelum byte; contoh, sebuah deret 9 bit akan ditampilkan sebagai {01}{1b} (FIPS 197, 2001).

2.5.3 *Array bytes*

Array bytes (*array* yang berisi sejumlah byte) direpresentasikan pada bentuk berikut :

$$a_0 a_1 a_2 .... a_{15}$$

Urutan byte-byte dan bit dalam bytes didapatkan dari sebuah rangkaian input berukuran 128 bit

$input_0 input_1 input_2 .....input_{126} input_{127}$ sebagai berikut:

$$a_0 = \{input_0 input_1 .....input_7\};$$
$$a_1 = \{input_8 input_9 .....input_{15}\};$$
$$\vdots$$
$$a_{15} = \{input_{120} input_{121} .....input_{127}\}.$$

Pola tersebut dapat diperluas menjadi rangkaian yang lebih panjang (contoh: untuk *key* dengan ukuran 192 dan 256 bit), sehingga bentuk umum dari pola rangkaian akan menjadi:

$$a_n = input_{8n}, input_{8n+1}, ..., input_{8n+7}$$ (FIPS 197, 2001).

2.5.4 *State*

Pada dasarnya, operasi-operasi algoritma AES dilakukan pada sebuah *array bytes* dua dimensi yang disebut *state***.** *State* terdiri dari 4 baris byte, setiap baris terdiri dari Nb buah byte, dimana Nb adalah panjang blok dibagi dengan 32. Pada *array state* yang dinotasikan dengan simbol S, setiap byte tunggal memiliki dua buah parameter, dimana nomor baris r berada pada *range* $0 \le r < 4$ dan nomor kolom c berada pada *range* $0 \le c < Nb$. Hal ini memungkinkan sebuah byte tunggal dari *state* untuk dapat dinyatakan sebagai $S_{r,c}$ atau S[*r,c*]. Sebagai standar yang digunakan di sini, Nb = 4, yang berarti jumlah kolom c adalah pada range $0 \le c < 4$ (Munir, 2006).

Pada awal proses enkripsi (*cipher*) dan dekripsi (*inverse cipher*), input – yaitu *array bytes* in_0, in_1, … in_{15} – disalin ke dalam *array state* sebagaimana yang diilustrasikan pada Gambar 2.12. Operasi-operasi *cipher* dan *invers cipher* kemudian dilakukan pada *array state* tersebut, sehingga didapat nilai akhir yang kemudian disalin ke dalam output – *array bytes* out_0, out_1, … out_{15} (FIPS 197, 2001).

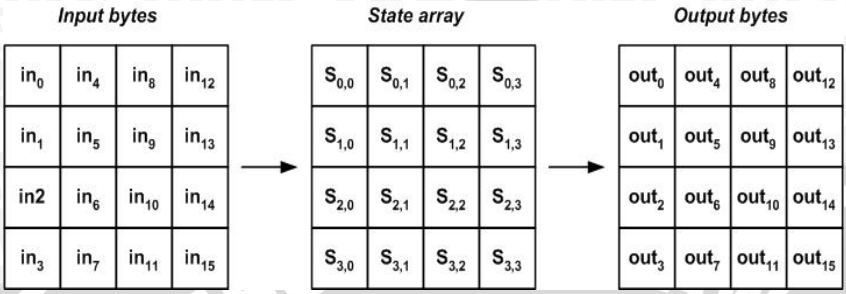
Dengan demikian, pada proses awal dari enkripsi (*cipher*) dan dekripsi (*inverse cipher*), *array input* , *in*, disalin ke dalam *array state* dengan skema sebagai berikut

S[r,c] ← *in*[r+4c] untuk $0 \le r < 4$ dan $0 \le c \le Nb$

dan pada proses akhir enkripsi dan dekripsi, *state* disalin ke dalam *array* output *out* sebagai berikut :

out[r+4c] ← S[r,c] untuk $0 \le r < 4$ dan $0 \le c \le Nb$

(Munir, 2006).



Gambar 2.12 *State array*, input dan output
(FIPS 197, 2001)

2.5.5 Pendahuluan matematika

Semua byte dalam AES diinterpretasikan sebagai elemen *finite field* menggunakan notasi pada 2.5.2. Elemen-elemen ini dapat dijumlah dan dikalikan namun operasinya berbeda dengan penjumlahan atau perkalian pada angka-angka biasa.

2.5.5.1 Penjumlahan

Penjumlahan dua elemen dalam *finite field* diperoleh dengan "menambah" koefisien-koefisien dari pangkat-pangkat yang sesuai dari dua elemen tersebut. Penjumlahan dilakukan dengan operasi XOR. Oleh karena itu, pengurangan dari polinomial identik dengan penjumlahannya.

Secara elternatif, penjumlahan elemen *finite field* dapat dijelaskan sebagai penjumlahan modulo 2 dari bit-bit yang terkait di dalam byte. Untuk 2 byte $\{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0\}$ dan $\{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0\}$, jumlahnya adalah $\{c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0\}$ dimana tiap-tiap $c_i = a_i \oplus b_i$. Contoh, ekspresi berikut ekuivalen satu sama lain:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$ (notasi polinomial)

$\{01010111\} \oplus \{10000011\} = \{11010100\}$ (notasi biner)

$\{57\} \oplus \{83\} = \{d4\}$ (notasi heksadesimal)

(FIPS 197, 2001)

2.5.5.2 Perkalian

Dalam representasi polinomial, perkalian dalam GF(2^8) (disimbolkan dengan $\bullet$) adalah perkalian dari polinomial modulo sebuah polinomial yang *irreducible* dengan derajat 8. Sebuah polinomial dikatakan *irreducible* jika satu-satunya pembaginya adalah 1 dan dirinya sendiri. Untuk AES, polinomial *irreducible* ini adalah:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

atau {01}{1b} dalam notasi heksadesimal. Sebagai contoh:

$\{57\} \bullet \{83\} = \{c1\}$ karena

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x +$$
$$x^6 + x^4 + x^2 + x + 1$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

dan

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \bmod x^8 + x^4 + x^3 + x + 1$$
$$= x^7 + x^6 + 1.$$

Reduksi modular oleh m(x) memastikan bahwa hasilnya akan selalu polinomial biner dengan derajat kurang dari 8, sehingga dapat direpresentasikan dalam sebuah byte (Gladman, 2001).

2.5.5.3 Perkalian menggunakan tabel

Jika elemen *finite field* tertentu (disebut generator) berulang-ulang dikalikan untuk menghasilkan sebuah deret dari pangkatnya, g^n, mereka secara progresif menghasilan semua 255 elemen tidak nol di dalam *field*. Ketika n mencapai 256 elemen *field* awal kembali muncul dengan g^{255} sehingga setara dengan {01}. Nilai n untuk tiap elemen *field* dapat dianggap sebagai logaritma dan ini memberikan jalan untuk mengubah perkalian menjadi penjumlahan. Maka, dua elemen $a = g^{\alpha}$ dan $b = g^{\beta}$ menghasilkan $a \bullet b = g^{\alpha+\beta}$. Dengan sebuah tabel logaritma yang menunjukkan pangkat dari generator untuk tiap elemen *finite field,* akan diperoleh pangkat α dan β yang

sesuai untuk elemen a dan b, dan menjumlahkan nilai ini untuk menemukan pangkat dari g untuk memperoleh hasil. Tabel kebalikan dapat dipakai untuk mencari elemen produk.

Karena dua nilai pangkat awal dapat mencapai 255, jumlahnya bisa jadi lebih besar dari 255, namun jika ini terjadi bisa dikurangkan nilai 255 dari nilainya sehingga membawanya kembali ke dalam *range* dari tabel karena $g^{255} = \{01\}$. Semua eksponen ditulis dalam heksadesimal (Gladman, 2001).

Tabel 2.11 Nilai-nilai N sehingga $\{xy\}=\{03\}^n$ untuk sebuah elemen *finite field* {xy} (Gladman, 2001)

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

Untuk *field* yang dipakai di dalam Rijndael, {03} adalah generator yang menghasilkan Tabel 2.11 dan Tabel 2.12. Dengan menggunakan contoh sebelumnya, Tabel 2.11 menunjukkan bahwa $\{57\}=\{03\}^{(62)}$ dan $\{83\}=\{03\}^{(50)}$, dimana tanda kurung dalam nilai eksponen menunjukkan bahwa itu adalah nilai heksadesimal. Ini memberi produk $\{57\} \bullet \{83\}=\{03\}^{(62+50)}$, dan karena (62)+(50)=(b2) dalam heksadesimal, Tabel 2.12 memberi hasil produk {c1}, seperti contoh sebelumnya. Kedua tabel tersebut dapat juga digunakan untuk menemukan *invers* dari sebuah elemen *field* karena $g^{(x)}$ memiliki sebuah invers yang berupa $g^{(ff)-(x)}$. Maka elemen $\{af\}=\{03\}^{(b7)}$ memiliki invers $g^{(ff)-(b7)} = g^{(48)} = \{62\}$. Semua elemen kecuali {00} memiliki invers (Gladman, 2001).

Tabel 2.12 Elemen *field* {E} sehingga {E}={03}^{xy} jika diketahui pangkat (xy) (Gladman, 2001)

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

2.5.6 Algoritma kriptografi AES

Algoritma kriptografi AES (*Advance Encryption Standard*), yang sering juga disebut dengan algoritma kriptografi Rijndael, sama seperti algoritma kriptografi DES yang menggunakan substitusi, permutasi dan sejumlah putaran (*cipher* berulang). Setiap putaran menggunakan kunci internal yang berbeda, dimana kunci pada setiap putaran disebut dengan *round key* (Munir, 2006).

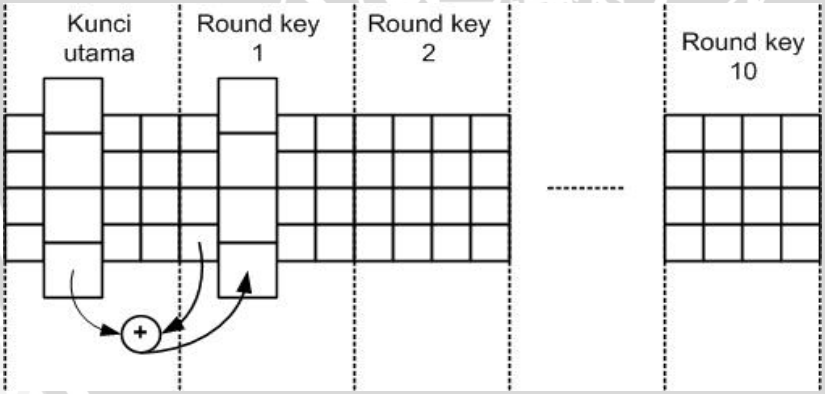
Algoritma kriptografi AES termasuk dalam klasifikasi algoritma kriptografi kunci simetri, kunci yang digunakan untuk enkripsi sama dengan kunci yang digunakan untuk dekripsi. Berbeda dengan DES, yang berorientasi bit, Rijndael beroperasi dalam byte. Algoritma AES dapat bekerja dalam tiga macam ukuran yakni 128 bit (16 byte), 192 bit (24 byte) dan 256 bit (32 byte) tabel 2.13. Namun dalam pengerjaan skripsi ini yang digunakan adalah AES 128 bit (Kurniawan, 2006).

2.5.7 Proses inisialisasi kunci internal

Dalam algoritma kriptografi AES terdapat 11 kali iterasi, dimana masing-masing iterasi akan menggunakan kunci yang berbeda. Untuk itu diperlukan 10 kunci internal yang berbeda (K_1, K_2, ..., K_{10}). Kunci internal ini diturunkan dari kunci eksternal yang berukuran 128 bit (*array* 4 x 4 dengan masing-masing elemen sebesar 1 byte). Kunci ini terdiri atas kunci utama (kunci eksternal yang diinputkan oleh user) dan 10 buah kunci putaran (*round key*) (Munir, 2006).

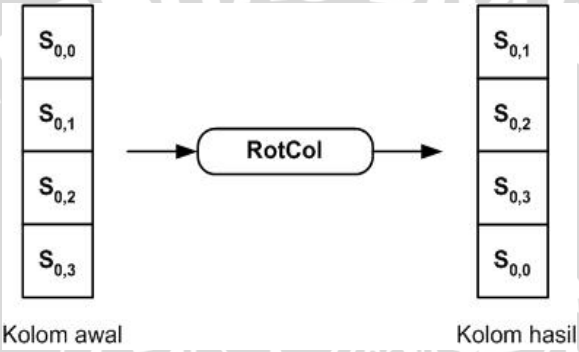
Proses inisialisasi kunci internal pada algoritma kriptografi AES terdiri dari dua bagian, yaitu proses untuk memperoleh nilai elemen-elemen pada kolom kedua hingga kolom keempat untuk masing-masing *round key* dan proses untuk memperoleh nilai elemen-elemen pada kolom pertama (Kurniawan, 2007).

Gambar 2.13 menunjukkan proses untuk memeperoleh nilai elemen-elemen pada kolom kedua hingga keempat. Untuk kolom kedua hingga keempat dari masing-masing *round key*, elemen-elemennya diperoleh dari hasil XOR antara satu kolom sebelumnya dengan kolom yang sama pada *array* kunci sebelumnya.



Gambar 2.13 Skema proses inisialisasi kunci pada AES (Kurniawan, 2007)

Untuk kolom pertama pada masing-masing *round key*, elemen-elemennya diperoleh melalui tiga transformasi yang dilaksanakan secara berurutan, yaitu RotCol, SubBytes dan XOR. Transformasi RotCol adalah operasi perputaran elemen berdasarkan barisnya, baris pertama menjadi baris keempat, baris kedua menjadi baris pertama, baris ketiga menjadi baris kedua, dan baris keempat menjadi baris ketiga (Stallings, 2005). Dapat dilihat pada gambar 2.14.



Gambar 2.14 Skema proses RotCol pada AES
(Munir, 2006)

Hasil dari transformasi RotCol tersebut menjadi masukan untuk transformasi SubBytes, yakni operasi substitusi yang dilakukan terhadap masing-masing byte dengan nilai yang ditunjukkan pada matriks substitusi S-Box. Transformasi SubBytes dalam proses inisialisasi kunci internal pada AES ini sama dengan transformasi SubBytes pada proses enkripsi (Stallings, 2005).

Hasil dari transformasi SubBytes akan di-XOR-kan dengan kolom pertama pada kunci sebelumnya dan Rcon untuk *round key* tersebut. Rcon adalah *array* 4 x 1 yang merupakan deretan konstanta yang digunakan dalam proses menghasilkan *round key*, dapat dilihat pada tabel 2.14. Dari proses inisialisasi kunci internal akan diperoleh 11 kunci internal yang diperoleh dari kunci eksternal yang diinputkan oleh user dan 10 *round key*. Ukuran untuk setiap kunci internal sebesar 128 bit dengan format *array* 4 x 4 dengan masing-masing elemen sebesar 1 byte (Kurniawan, 2007).

Tabel 2.13 Tabel Rcon untuk 10 *round key* AES (Kurniawan, 2007)

Rcon 1	Rcon 2	Rcon 3	Rcon 4	Rcon 5	Rcon 6	Rcon 7	Rcon 8	Rcon 9	Rcon 10
01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

2.5.8 Contoh proses inisialisasi kunci internal

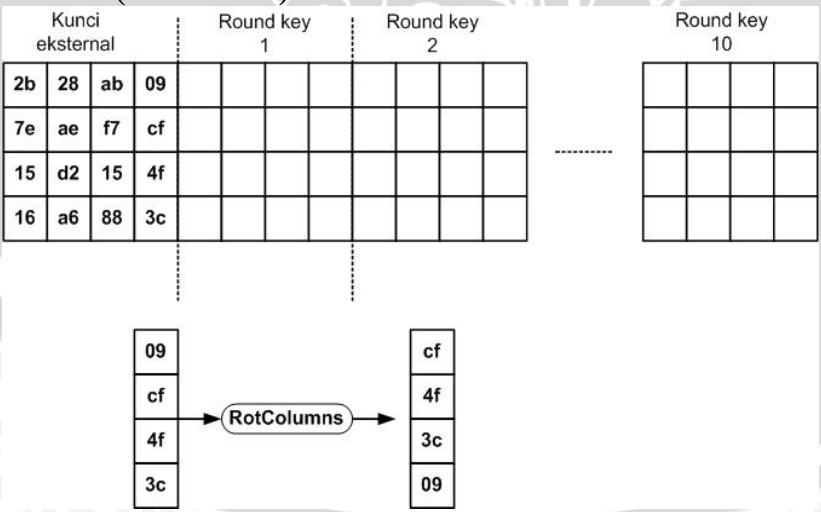
Berikut ini adalah contoh tahapan dalam proses inisialisasi kunci internal:

1) Misalkan *array* kunci eksternal (Gambar 2.15).

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

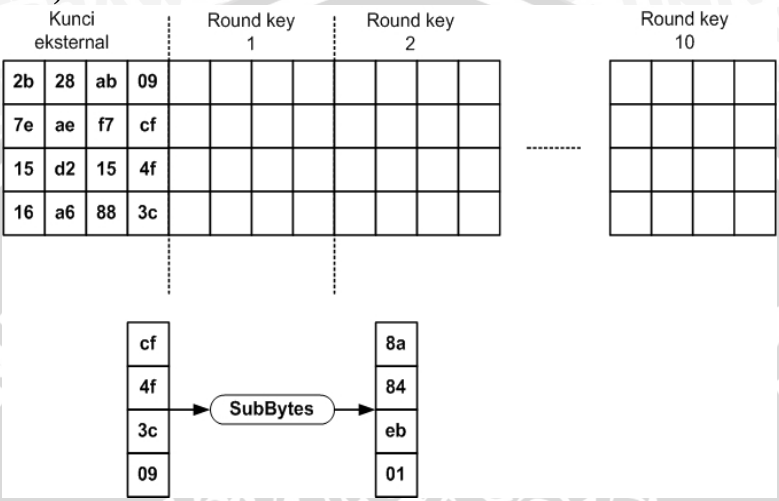
Gambar 2.15 Contoh *array* kunci eksternal

2) Untuk memperoleh kolom pertama pada *round key* 1, dilakukan transformasi RotCol pada kolom terakhir dari *array* kunci eksternal (Gambar 2.16).



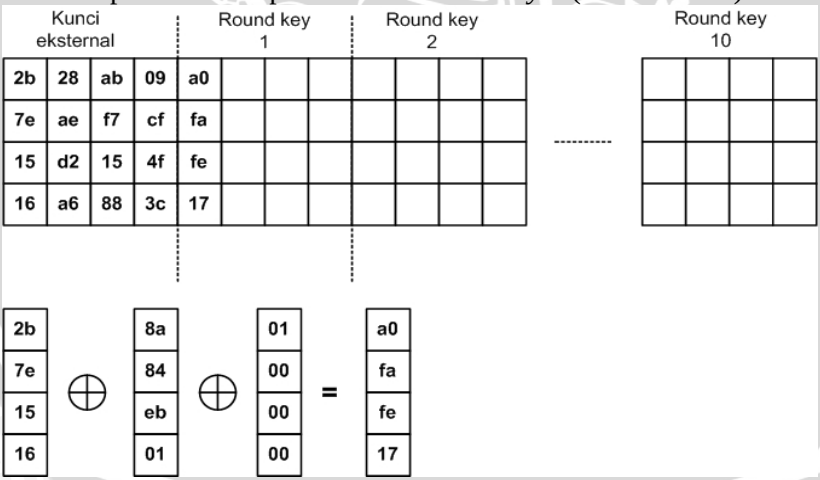
Gambar 2.16 Contoh transformasi RotCol

3) Transformasi SubBytes pada kolom yang terakhir *array* kunci eksternal yang sudah dilakukan transformasi RotCol (Gambar 2.17).



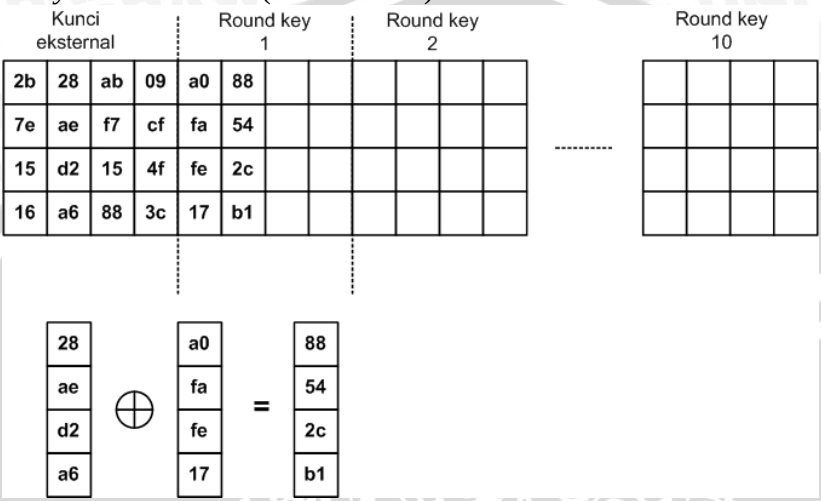
Gambar 2.17 Contoh transformasi SubBytes

4) Kolom ke-1 *array* kunci eksternal di-XOR-kan dengan kolom hasil SubBytes tersebut dan Rcon1, dan kemudian hasilnya merupakan kolom pertama dari *round key* 1 (Gambar 2.18).



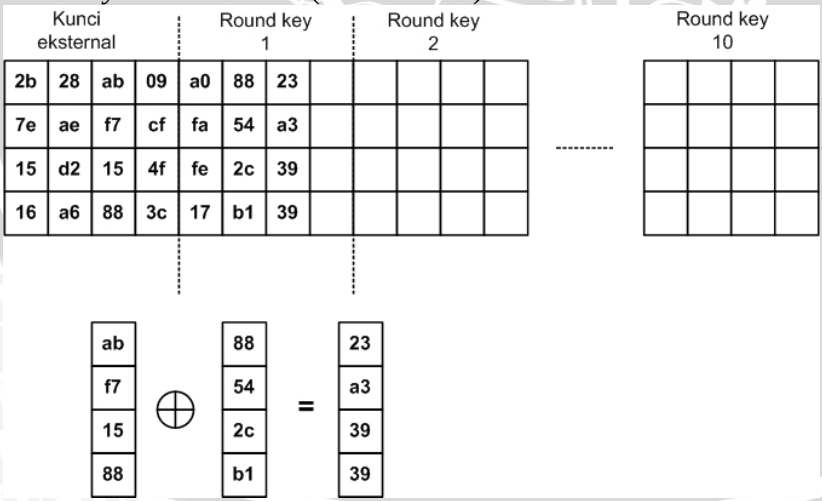
Gambar 2.18 Contoh cara memperoleh kolom pertama *round key* 1

5) Untuk memperoleh kolom kedua dari *round key* 1, dilakukan peng-XOR-an kolom pertama *round key* 1 dengan kolom kedua *array* kunci eksternal (Gambar 2.19).



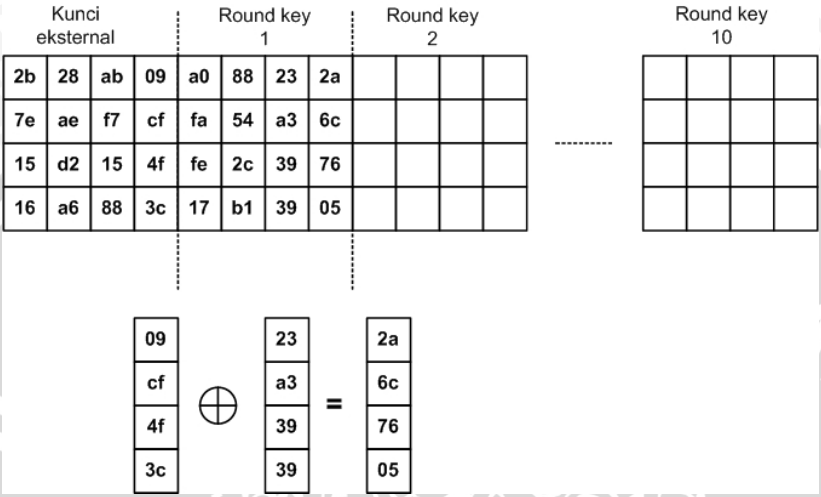
Gambar 2.19 Contoh cara memperoleh kolom kedua *round key* 1

6) Untuk memperoleh kolom ketiga dari *round key* 1, dilakukan peng-XOR-an kolom kedua *round key* 1 dengan kolom ketiga *array* kunci eksternal (Gambar 2.20).



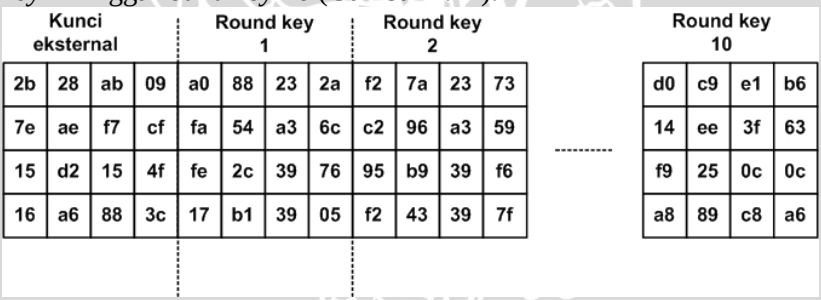
Gambar 2.20 Contoh cara memperoleh kolom ketiga *round key* 1

7) Untuk memperoleh kolom keempat dari *round key* 1, dilakukan peng-XOR-an kolom ketiga *round key* 1 dengan kolom keempat *array* kunci eksternal (Gambar 2.21).



Gambar 2.21 Contoh cara memperoleh kolom keempat *round key* 1

8) Langkah 2 hingga 7 diulang 9 kali lagi untuk memperoleh *round key* 2 hingga *round key* 10 (Gambar 2.22).



Gambar 2.22 Kunci eksternal dan *round key*

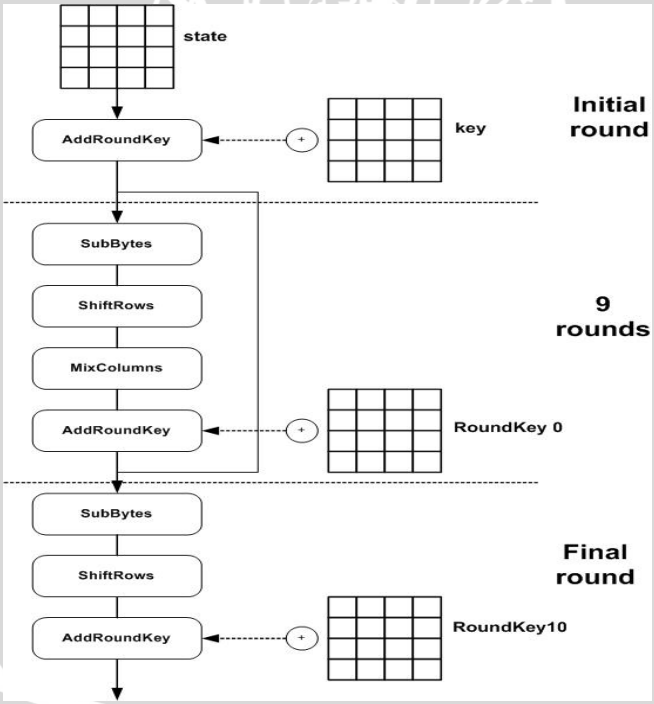
2.5.9 Proses enkripsi

Garis besar algoritma kriptografi AES yang beroperasi pada blok 128 bit dengan kunci 128 bit adalah sebagai berikut:

1. *Initial Round*. Dalam proses ini terdapat proses *AddRoundKey*, yakni melakukan XOR antara state awal (*plaintext*) dengan kunci (*cipher key*).

2. Putaran sebanyak Nr-1 kali. Proses yang dilakukan pada setiap putaran adalah:
   a. *SubBytes*: melakukan substitusi byte-byte dalam state dengan menggunakan tabel substitusi kotak-S (S-Box).
   b. *ShiftRows*: melakukan pergeseran baris-baris *array state* secara *wrapping.*
   c. *MixColumns*: mengacak byte-byte di masing-masing kolom *array* state.
   d. *AddRoundKey*: melakukan XOR antara state sekarang dengan kunci *round key*.
3. *Final round*: proses untuk putaran terakhir. Dalam proses ini terdapat 3 proses yang dilakukan secara berurutan, yakni:
   a. *SubBytes*
   b. *ShiftRows*
   c. *AddRoundKey* (Munir, 2006).

Skema proses enkripsi AES dapat dilihat pada gambar 2.23.



Gambar 2.23 Skema proses enkripsi AES
(Kurniawan, 2007)

Proses enkripsi dalam algoritma kriptografi AES dilakukan sebanyak 11 kali, satu kali dalam *initial round*, 9 kali dalam *round(s)* dan 1 kali *final round*.

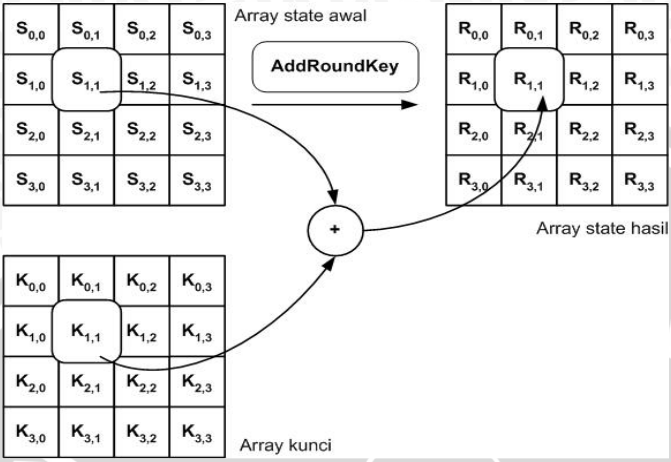
Algoritma Rijndael memiliki 3 parameter:

1. *plaintext*: *array* yang berukuran 16 byte, yang berisi data masukan.
2. *ciphertext*: *array* yang berukuran 16 byte, yang berisi hasil enkripsi.
3. *key*: *array* yang berukuran 16 byte, yang berisi kunci *ciphering* (disebut juga *cipher key*) (Munir, 2006).

Dengan 16 byte, maka baik blok data dan kunci yang berukuran 128 bit dapat disimpan dalam ketiga *array* tersebut (128 = 16 x 8). Selama kalkulasi *plaintext* menjadi *ciphertext*, status dari data sekarang disimpan di dalam *array of bytes* dua dimensi, state, yang berukuran nrows x ncols. Untuk blok data 128 bit, ukuran state adalah 4x4. Elemen *array state* diacu sebagai S[r,c], dengan $0 \leq c \leq$ Nb (Nb adalah panjang blok dibagi 32. pada AES-128, Nb = 128/32 = 4) (FIPS 197, 2001).

Dalam *initial round*, transformasi *AddRoundKey()* dilakukan terhadap kunci utama. Sedangkan dalam 10 *round* yang lain, proses *AddRoundKey* dilakukan terhadap kunci putaran (*round key*). Proses *AddRoundKey* didefinisikan sebagai operasi XOR antara *array state* dengan *round key*. Operasi XOR dilakukan pada masing-masing byte dalam *array* sehingga menghasilkan nilai baru pada *array* hasil dengan ukuran *array* hasil sama dengan ukuran *array state* awal dan *array key*, yaitu sebesar 4x4. Hasil untuk masing-masing baris dan kolom pada *array state* hasil diperoleh dari hasil operasi XOR antara *array state* awal dengan *array key* untuk baris dan kolom yang sama. Ilustrasi dari proses *AddRoundKey* dapat dilihat pada Gambar 2.16 (Kurniawan, 2007).

Transformasi *SubBytes()* memetakan setiap byte dari *array state* dengan menggunakan tabel substitusi S-Box. Tidak seperti DES yang mempunyai S-Box berbeda pada setiap putaran, AES hanya mempunyai satu buah S-Box. Tabel S-Box dapat dilihat pada Tabel 2.15 (FIPS 197, 2001).

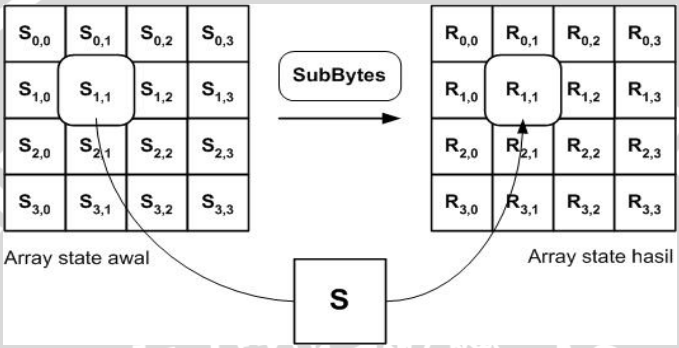


Gambar 2.24 Transformasi *AddRoundKey* (www.wikipedia.org, 2008)

Tabel 2.14 Tabel S-Box AES (FIPS 197, 2001)

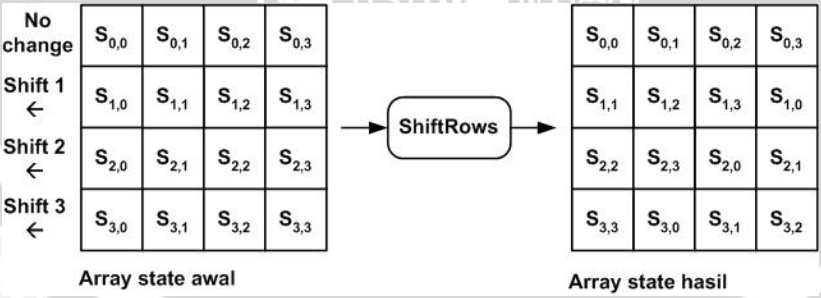
HEX		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29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

Cara pensubstitusian adalah sebagai berikut: untuk setiap byte pada *array*, misalkan S[r,c] = xy, yang dalam hal ini xy adalah digit heksadesimal dari nilai S[r,c], maka nilai substitusinya, yang dinyatakan dengan S'[r,c], adalah elemen di dalam S-Box yang merupakan perpotongan baris x dengan kolom y. Gambar 2.25 menunjukkan transformasi SubBytes (Munir, 2006).



Gambar 2.25 Transformasi *SubBytes*
(www.wikipedia.org, 2008)

Transformasi *ShiftRows()* melakukan pergeseran secara *wrapping* (siklik) pada 3 baris terakhir dari *array state*. Jumlah pergeseran bergantung pada nilai baris (r). Baris r = 1 digeser sejauh 1 byte, baris r = 2 digeser sejauh 2 byte, dan baris r = 3 digeser sejauh 3 byte. Baris r = 0 tidak digeser. Gambar 2.26 memperlihatkan transformasi *ShiftRows* (Munir, 2006).



Gambar 2.26 Transformasi *ShiftRows*
(www.wikipedia.org, 2008)

Transformasi *MixColumns()* dilakukan oleh algoritma Rijndael, setelah transformasi *ShiftRows*, merupakan sumber utama dari difusi pada Rijndael (www.wikipedia.org, 2008). Difusi merupakan prinsip yang menyebarkan pengaruh satu bit *plaintext* atau kunci ke sebanyak mungkin *ciphertext*. Sebagai contoh, pengubahan kecil pada *plaintext* sebanyak satu atau dua bit menghasilkan perubahan pada *ciphertext* yang tidak dapat diprediksi. Prinsip difusi juga menyembunyikan hubungan statistik antara *plaintext, ciphertext* dan kunci sehingga membuat kriptanalisis menjadi sulit (Munir, 2006).

Transformasi *MixColumns()* mengalikan setiap kolom dari *array state* dengan polinom $a(x) \bmod (x^4 + 1)$. Setiap kolom diperlakukan sebagai polinom 4 suku pada $GF(2^8)$. Polinom $a(x)$ yang ditetapkan pada persamaan 2.11

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2.11)$$

Transformasi ini dinyatakan sebagai perkalian matriks seperti pada persamaan 2.12

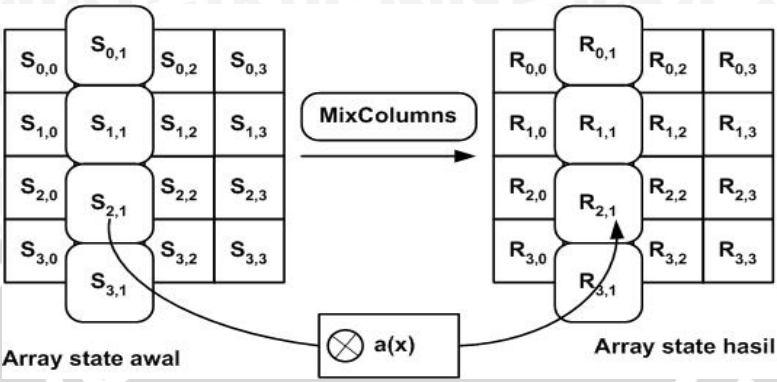
$$s'(x) = a(x) \otimes s(x) \quad (2.12)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Hasil dari perkalian matriks tersebut, setiap byte dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap byte tersebut pada persamaan 2.13

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \otimes (\{03\} \bullet s_{1,c}) \otimes s_{2,c} \otimes s_{3,c} \\ s'_{1,c} &= s_{0,c} \otimes (\{02\} \bullet s_{1,c}) \otimes (\{03\} \bullet s_{2,c}) \otimes s_{3,c} \\ s'_{2,c} &= s_{0,c} \otimes s_{1,c} \otimes (\{02\} \bullet s_{2,c}) \otimes (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \otimes s_{0,c} \otimes s_{1,c} \otimes (\{02\} \bullet s_{3,c}) \end{aligned} \quad (2.13)$$

Gambar 2.27 memperlihatkan transformasi *MixColumns* (FIPS 197, 2001).



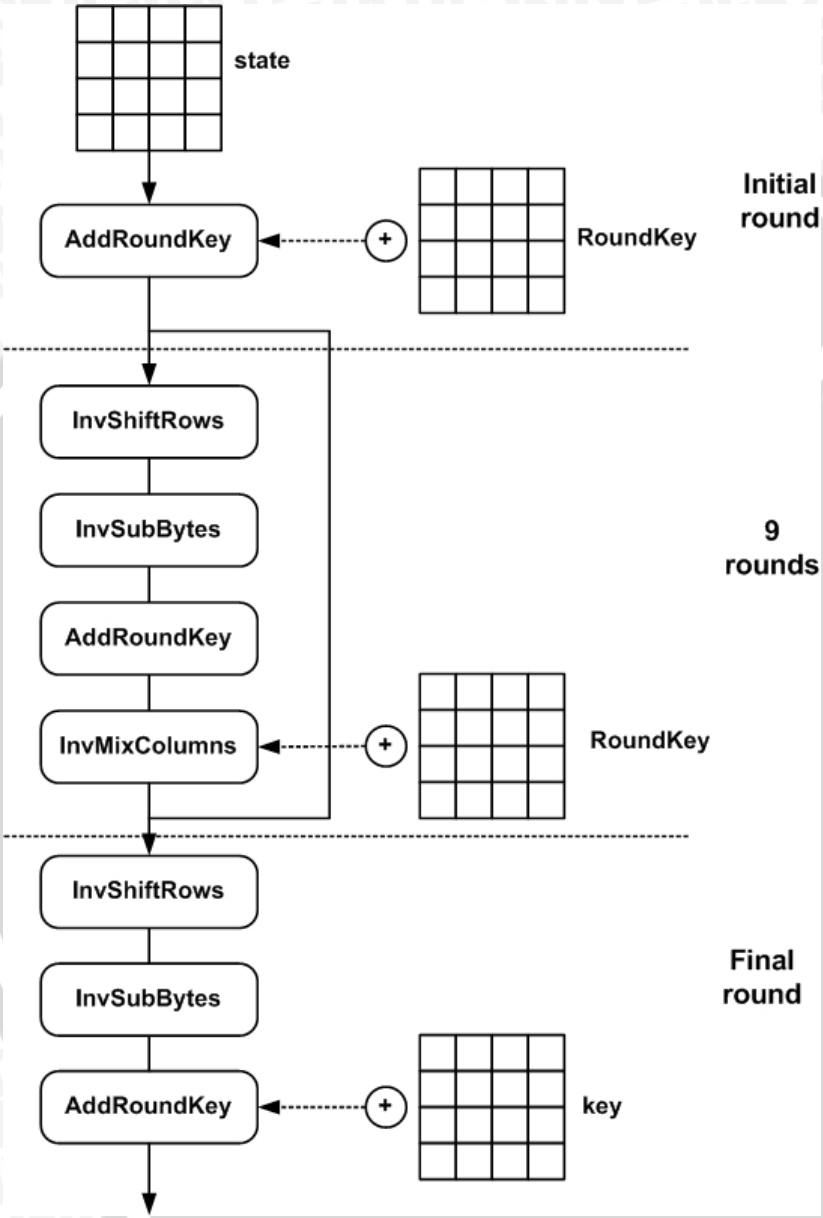
Gambar 2.27 Transformasi *MixColumns*
(www.wikipedia.org, 2008)

2.5.10 Proses dekripsi

Proses dekripsi dalam algoritma AES (Rijndael) merupakan rangkaian pembalikan dari proses-proses yang dilakukan dalam proses enkripsinya. Gambar 2.28 menunjukkan skema global proses dekripsi AES.

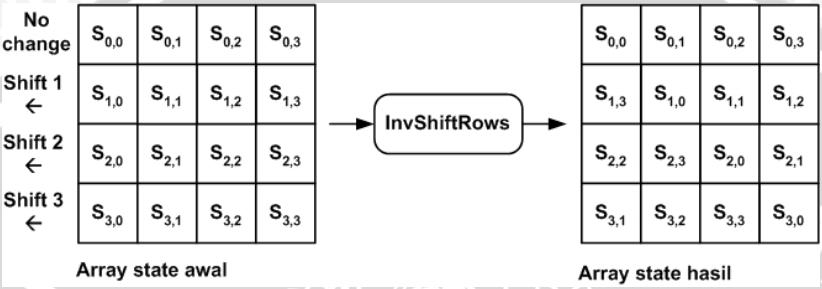
Transformasi *AddRoundKey* tidak mengalami pembalikan (invers) karena proses *AddRoundKey* merupakan operasi XOR. Pembalikan dalam proses ini terletak pada urutan kunci yang digunakan. Jika pada proses enkripsi, urutan kunci internal yang digunakan adalah kunci eksternal (kunci utama), *round key* 1, *round key* 2, *round key* 3, dan seterusnya hingga *round key* 10 (K_0, K_1, K_2, K_3, ..., K_{10}); maka pada proses dekripsi urutan kunci yang digunakan adalah kebalikannya yanki menjadi *round key* 10, *round key* 9, *round key* 8, dan seterusnya sampai *round key* 1, kemudian diakhiri dengan kunci eksternal (K_{10}, K_9, K_8, ..., K_0) (Kurniawan, 2007).

Transformasi *InvShiftRows* didefinisikan sebagai operasi pergeseran byte-byte yang terletak pada masing-masing baris. Pembalikan dalam transformasi ini terletak pada bentuk pergeserannya. Jika pada proses enkripsi, transformasi *ShiftRows* melakukan pergeseran ke kiri, maka pada proses dekripsi transformasi *InvShiftRows* melakukan pergeseran ke kanan (FIPS 197, 2001).



Gambar 2.28 Skema global proses dekripsi AES
(Kurniawan, 2007)

Banyaknya pergeseran yang dilakukan sama dengan banyaknya pergeseran dalam transformasi *ShiftRows,* yakni baris ke-0 digeser sejauh 0 kolom, baris ke-1 digeser sejauh 1 kolom, baris ke-2 digeser sejauh 2 kolom dan pada baris ke-3 digeser sejauh 3 kolom. Gambar 2.29 menunjukkan transformasi *InvShiftRows* (Kurniawan, 2007).



Gambar 2.29 Transformasi *InvShiftRows* (Kurniawan, 2007)

Transformasi *InvSubBytes* didefinisikan sebagai operasi substitusi yang dilakukan pada masing-masing byte dalam *array state*. Masing-masing byte dalam *array state* idsubstitusi dengan nilai baru sesuai dengan nilai yang ditunjukkan dalam matriks substitusi S-Box balikan (*InvS-Box*). Tabel *InvS-Box* ditunjukkan dalam Tabel 2.16.

Transformasi *InvMixColumns* (Gambar 2.30) didefinisikan sebagai operasi yang dilakukan terhadap byte-byte yang terletak pada masing-masing kolom. Operasi yang dilakukan terhadap masing-masing kolom adalah operasi perkalian matriks, sama seperti *MixColumns.* Hal yang membedakan antara transformasi *MixColumns* dan *InvMixColumns* adalah matriks 4x4 yang menjadi matriks pengalinya.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Gambar 2.30 Transformasi *InvMixColumns*

Hasil dari perkalian matriks tersebut, setiap byte dalam kolom *array state* akan digantikan dengan nilai baru. Persamaan matematis untuk setiap byte tersebut pada persamaan 2.14

$$
\begin{aligned}
s'_{0,c} &= (\{0e\} \bullet s_{0,c}) \otimes (\{0b\} \bullet s_{1,c}) \otimes (s_{2,c}) \otimes s_{3,c} \\
s'_{1,c} &= s_{0,c} \otimes (\{02\} \bullet s_{1,c}) \otimes (\{03\} \bullet s_{2,c}) \otimes s_{3,c} \\
s'_{2,c} &= s_{0,c} \otimes s_{1,c} \otimes (\{02\} \bullet s_{2,c}) \otimes (\{03\} \bullet s_{3,c}) \\
s'_{3,c} &= (\{03\} \bullet s_{0,c}) \otimes s_{0,c} \otimes s_{1,c} \otimes (\{02\} \bullet s_{3,c})
\end{aligned}
\qquad (2.14)
$$

(FIPS 197, 2001).

Tabel 2.15 *InvS-Box* (FIPS 197, 2001).

HEX		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	D5	30	36	A5	38	Bf	40	A3	9e	81	F3	D7	Fb
	1	7c	E3	39	82	9b	2f	Ff	87	34	8e	43	44	C4	De	E9	Cb
	2	54	7b	94	32	A6	C2	23	3d	Ee	4c	95	0b	42	Fa	C3	4e
	3	08	2e	A1	66	28	D9	24	B2	76	5b	A2	49	6d	8b	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5c	Cc	5d	65	B6	92
	5	6c	70	48	50	Fd	Ed	B9	Da	5e	15	46	57	A7	8d	9d	84
	6	90	D8	Ab	00	8c	Bc	D3	0a	F7	E4	58	05	B8	B3	45	06
	7	d0	2c	1e	8f	Ca	3f	0f	02	C1	Af	Bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	Dc	Ea	97	F2	Cf	Ce	F0	B4	E6	73
	9	96	Ac	74	22	E7	Ad	35	85	E2	F9	37	E8	1c	75	Df	6e
	a	47	F1	1a	71	1d	29	C5	89	6f	B7	62	0e	Aa	18	Be	1b
	b	fc	56	3e	4b	C6	D2	79	20	9a	Db	C0	Fe	78	Cd	5a	F4
	c	1f	Dd	A8	33	88	07	C7	31	B1	12	10	59	27	80	Ec	5f
	d	60	51	7f	A9	19	B5	4a	0d	2d	E5	7a	9f	93	C9	9c	Ef
	e	A0	E0	3b	4d	Ae	2a	F5	B0	C8	Eb	Bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	D6	26	E1	69	14	63	55	21	0c	7d

2.6 Avalanche Effect

Salah satu karakteristik untuk menentukan baik atau tidaknya suatu algoritma kriptografi adalah dengan melihat nilai *avalanche effect*-nya, yakni perubahan kecil pada *plaintext* maupun *key* akan menyebabkan perubahan yang signifikan pada *ciphertext* yang dihasilkan, atau dengan kata lain perubahan 1 bit pada *plaintext* maupun *key* akan menghasilkan perubahan beberapa bit pada

ciphertext. Semakin banyak perubahan bit pada *ciphertext* maka semakin baik algoritma kriptografi tersebut.

Contoh menghitung nilai *avalanche effect* :
Plaintext 1: 00000000000000000000000000000000 (heksadesimal)
Plaintext 2: 80000000000000000000000000000000 (heksadesimal)
Key : 00000000000000000000000000000000 (heksadesimal)
Ciphertext yang dihasilkan:
Ciphertext 1: DBAD348CF30BBF8E64B5E5D3065D6898 (heksadesimal)
Ciphertext 2: D2734057410AE10710C4922DCC9B34FA (heksadesimal)

Ciphertext 1:
11011011101011000110100100011001111001100001011101111111000111001100100101101011110010111010011000001100101110101110100010011000 (biner)
Ciphertext 2:
11010010011100110100000001010111010000010000101011100001000001110001000011000100100100100010110111001100100110110011010011111010 (biner)

Perbedaan bit pada *ciphertext* 1 dan *ciphertext* 2 sebanyak 67 bit dari total 128 bit *ciphertext* atau sekitar 52% (Budiyono, 2004).


UNIVERSITAS BRAWIJAYA

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab ini akan dibahas mengenai metode dan tahap-tahap yang digunakan dalam pembuatan perangkat lunak *avalanche effect.* Diagram alir pembuatan perangkat lunak dapat dilihat pada gambar 3.1. Adapun tahapan pembuatannya adalah sebagai berikut:

1. Melakukan studi literatur mengenai algoritma kriptografi DES, algoritma kriptografi AES dan *avalanche effect.*
2. Menganalisis dan merancang perangkat lunak *avalanche effect.*
3. Implementasi perangkat lunak berdasarkan analisis dan perancangan yang dilakukan.
4. Melakukan uji coba terhadap perangkat lunak.
5. Melakukan evaluasi hasil yang diperoleh dari uji coba perangkat lunak.

Gambar 3.1 Diagram alir pembuatan perangkat lunak

3.1 Deskripsi Umum Perangkat Lunak

Perangkat lunak yang akan dibuat merupakan implementasi dari algoritma kriptografi DES dan AES untuk proses enkripsi, yang kemudian menghitung nilai *avalanche effect* dari kedua algoritma tersebut. Tujuan dari pembuatan perangkat lunak ini adalah untuk mengetahui nilai *avalanche effect* dari algoritma DES dan AES, sehingga nantinya dapat diperoleh kesimpulan algoritma mana yang memilki nilai *avalanche effect* yang lebih baik.

Sebagaimana yang telah dituliskan pada sub bab 2.6, *avalanche effect* bertujuan untuk melihat seberapa besar pengaruh yang diakibatkan oleh perubahan kecil pada *plaintext atau key* terhadap *ciphertext.* Untuk itu dalam pengerjaan Skripsi ini dilakukan dua tipe uji coba yakni:

1) 1 *plaintext* diuji coba dengan menggunakan beberapa kunci eksternal yang berbeda.
2) Beberapa *plaintext* yang berbeda diuji coba dengan menggunakan 1 kunci eksternal.

Pada uji coba pertama, dalam menghasilkan kunci eksternal yang berbeda terlebih dahulu dibuat sebuah kunci eksternal yang terdiri dari karakter-karakter (huruf, angka atau simbol lainnya) dalam skripsi ini digunakan inputan kunci eksternal berupa biner. Kemudian dilakukan perubahan pada bit-bit kunci eksternal ini sebesar n > 1, dimana n adalah bilangan genap, agar diperoleh kunci eksternal yang lainnya. Hal ini bertujuan untuk melihat seberapa besarkah perubahan kecil yang diberikan pada kunci eksternal mempengaruhi *ciphertext* yang dihasilkan dari proses enkripsi. Sebagai contoh:

- ✓ kunci eksternal 1 = 00000000000000000000000000000000000000000000000000000000000000
- ✓ untuk memperoleh kunci eksternal yang lainnya, maka dilakukan perubahan pada bit-bit kunci eksternal 1 tersebut, dimulai dari bit paling kanan sebesar 2 bit, 4 bit, 6 bit dan seterusnya.
- ✓ Kunci eksternal 2 = 00000000000000000000000000000000000000000000000000000000000**11**
  Kunci eksternal 3 = 00000000000000000000000000000000000000000000000000000000**1111**

Kunci eksternal 4 =
00000000000000000000000000000000000000000000000000000000000000**111111**
dan seterusnya.

Pada uji coba yang kedua, sama halnya dengan uji coba pertama, dalam menghasilkan *plaintext* yang berbeda-beda terlebih dahulu dibuat sebuah *plaintext* yang berisi karakter-karakter. Kemudian dilakukan perubahan atau modifikasi pada bt-bit *plaintext* tersebut sebesar n > 1, dimana n adalah bilangan genap, agar diperoleh *plaintext* yang lainnya. Hal ini bertujuan untuk melihat seberapa besarkah perubahan kecil yang diberikan pada *plaintext* akan mempengaruhi *ciphertext* yang dihasilkan dari proses enkripsi. Sebagai contoh:

- ✓ *Plaintext 1* = ABCDEFGH
  Atau jika dikonversikan dalam biner menjadi =
  010000010100001001000011010001000100010101000110010001110100 1000
- ✓ untuk memperoleh *plaintext* yang lainnya, maka dilakukan perubahan pada bit-bit *plaintext* 1 tersebut, dimulai dari bit paling kanan sebesar 2 bit, 4 bit, 6 bit dan seterusnya.
- ✓ *Plaintext 2* =
  010000010100001001000011010001000100010101000110010001110100 1**11**
  Plaintext 3 =
  010000010100001001000011010001000100010101000110010001110100 **0111**
  Plaintext 4 =
  010000010100001001000011010001000100010101000110010001110**110111**
  dan seterusnya.

Setelah menentukan *plaintext* dan kunci eksternal yang digunakan untuk melakukan uji coba, maka dilakukan proses enkripsi terhadap *plaintex* tersebut dengan menggunakan kunci eksternal untuk memperoleh *ciphertext* dari setiap jenis uji coba. Pada uji coba pertama, misalkan *plaintext* yang digunakan adalah *PlaintextA, PlaintextA* dienkripsikan dengan menggunakan kunci eksternal 1, kunci eksternal 2, hingga kunci eksternal yang terakhir pada uji coba tersebut dan akan menghasilkan *Ciphertext A1, Ciphertext A2*, dan seterusnya hingga *CiphertextAx*. Nilai *avalanche effect* uji coba

pertama ini diperoleh dengan menghitung banyaknya jumlah bit yang berbeda dari *Ciphertext A2* (*Ciphertext* hasil enkripsi *Plaintext A* degan kunci eksternal 2), *Ciphertext A3*, hingga *Ciphertext Ax* terhadap *Ciphertext A1*(*Ciphertext* hasil enkripsi *Plaintext A* degan kunci eksternal 1).

Demikian pula halnya untuk uji coba kedua, misalkan kunci eksternal yang digunakan adalah KunciEksternalP, *plaintext1*, *plaintext2*, hingga *plaintextX* dienkripsikan dengan menggunakan KunciEksternalP sehingga menghasilkan *Ciphertext1P*, *Ciphertext2P*, *Ciphertext3P* dan seterusnya hingga *CiphertextXP*. Nilai *avalanche effect* uji coba kedua diperoleh dengan menghitung banyaknya jumlah bit berbeda dari *Ciphertext2P*, *Ciphertext3P* hingga *CiphertextXP* terhadap *Ciphertext1P*.

Untuk melakukan enkripsi dengan menggunakan algoritma kriptografi DES, tahap-tahapnya adalah:

1) User menginputkan kunci eksternal (*external key*) sebesar 64 bit biner (8 karakter) dan *plaintext* yang akan dienkripsikan pada perangkat lunak.
2) Kunci eksternal diproses seperti yang dijelaskan pada sub bab 2.4.2 sehingga diperoleh K_1, K_2, ... K_{16}.
3) Dilakukan IP terhadap blok *plaintext*, hasilnya dibagi menjadi dua bagian yakni kiri dan kanan (L_1 dan R_1) masing-masing sebesar 32 bit dan digunakan sebagai inputan *round 1*.
4) R_1 dan K_1 menjadi inputan fungsi F untuk *round 1* (seperti yang telah dijelaskan pada sub bab 2.4.3). Kemudian fungsi F di-XOR-kan dengan L_1 untuk menghasilkan R_2.
5) Tahap nomor 5 dilakukan juga terhadap *round 2* hingga *round 15*.
6) Pada *round 16*, R_{15} menjadi L_{16} dan R_{16} diperoleh dari L_{15} yang di-XOR-kan dengan fungsi F *round 15*.
7) R_{16} L_{16} digabungkan dan selanjutnya dilakukan IP^{-1}
8) Tahap nomor 8 menghasilkan *ciphertext*.

Sedangkan untuk melakukan enkripsi dengan menggunakan algoritma kriptografi AES, tahap-tahapnya adalah:

1) User menginputkan kunci eksternal (*external key*) sebesar 128 bit (16 karakter) dan *plaintext* yang akan dienkripsi pada perangkat lunak.

2) Kunci eksternal diproses seperti yang telah dijelaskan pada sub bab 2.5.7 sehingga menghasilkan 10 kunci internal (*round key*) K_1, K_2, ..., K_{10}.
3) *Plaintext* dibagi ke dalam blok-blok yang berukuran 16 byte dan disalin ke *array state* untuk dilakukan proses enkripsi.
4) Pada *initial round*, dilakukan transformasi *AddRoundKey* pada *state array plaintext*, yakni meng-XOR-kan *state array plaintext* dengan kunci eksternal.
5) Untuk round ke-1 hingga ke-9 dilakukan empat transformasi secara berurutan terhadap *state array* hasil proses sebelumnya yakni transformasi *SubBytes*, *ShiftRows*, *MixColumns* dan *AddRoundKey*.
6) Pada *final round* (putaran terakhit atau *round 10*), dilakukan tiga transformasi secara berurutan terhadap *state array round 9*, yakni *SubBytes*, *ShiftRows* dan *AddRoundKey* dan menghasilkan output berupa *ciphertext*.

Untuk

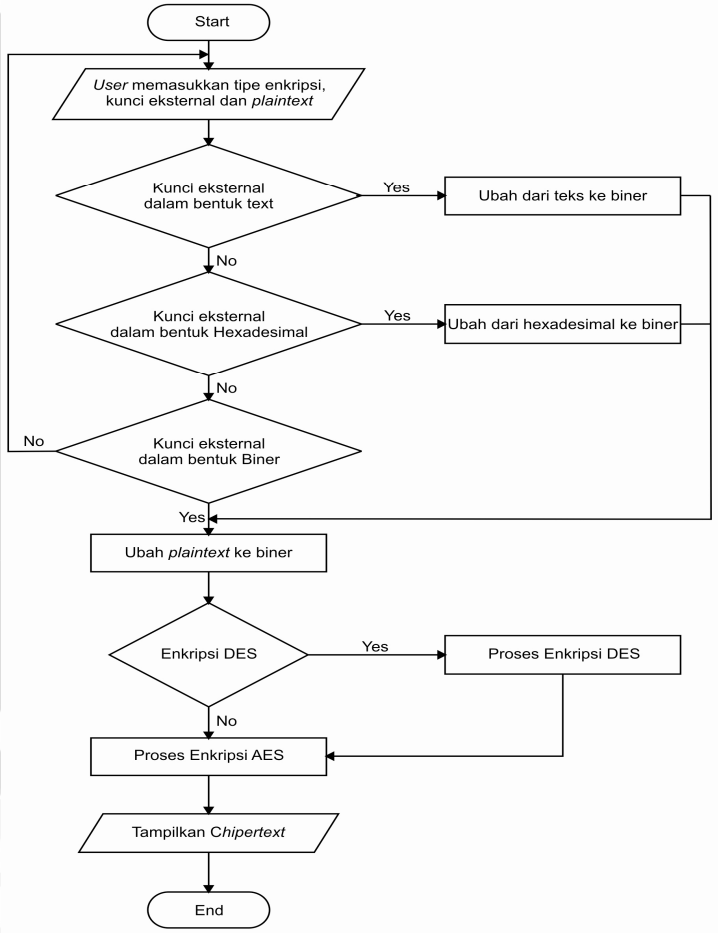
Perangkat lunak ini memiliki dua tahap, yakni tahap enkripsi *plaintext* dan tahap menghitung nilai *avalanche effect*. Pada tahap enkripsi (Gambar 3.2), proses-proses yang dilakukan adalah:

1) User memilih akan melakukan enkripsi dengan menggunakan algoritma DES atau AES terlebih dahulu.
2) Apabila user memilih untuk melakukan enkripsi dengan menggunakan algoritma kriptografi DES, maka perangkat lunak meminta inputan berupa *external key* sepanjang 8 karakter. Sedangkan untuk melakukan enkripsi dengan menggunakan algoritma AES, user perlu menginputkan *external key* sepanjang 16 karakter.
3) Perangkat lunak meminta inputan *plaintext* yang akan dienkripsi.
4) Perangkat lunak menghasilkan *ciphertext* yang kemudian digunakan sebagai inputan untuk tahap penghitungan nilai *avalanche effect*.

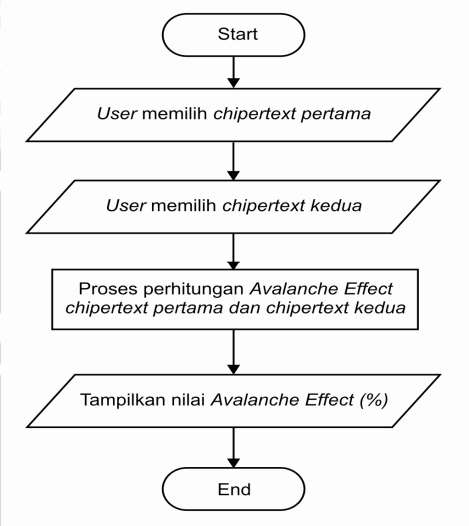
Untuk tahap penghitungan nilai *avalanche effect* (Gambar 3.3), proses yang dilakukan adalah:

1) User menginputkan dua buah *ciphertext* yang akan dibandingkan. Untuk uji coba pertama, user menginputkan *CiphertextA1* dan *CiphertextAx*. Sedangkan untuk uji coba kedua, user menginputkan *Ciphertext1P* dan *CiphertextXP*.

2) Perangkat lunak menghitung berapa banyak bit yang berbeda dari kedua *ciphertext* yang dibandingkan.
3) Perangkat lunak memberikan output berupa nilai *avalanche effect* yang diperoleh dengan cara melakukan pembagian jumlah bit yang berbeda antara *CiphertextA1* dan *CiphertextAx* atau *Ciphertext1P* dan *CiphertextXP* dengan total jumlah bit *CiphertextA1* atau *Ciphertext1P* tersebut. Dalam hal ini, jumlah bit yang dimiliki oleh *CiphertextA1* atau *Ciphertext1P* sama dengan jumlah bit yang dimiliki oleh *CiphertextAx* atau *CiphertextXP*.



Gambar 3.2 Flowchart Proses enkripsi



Gambar 3.3 Flowchart Proses perhitungan *avalanche effect*

3.2 Perancangan Proses

3.2.1 Parameter input perangkat lunak

Pada tahap enkripsi, ada beberapa parameter yang perlu diinputkan pada perangkat lunak, yakni:

1) algoritma yang akan digunakan untuk melakukan enkripsi, apakah dengan algoritma DES atau AES.
2) *external key* sebesar 64 bit biner atau 16 heksadesimal atau 8 karakter jika melakukan enkripsi dengan menggunakan algoritma kriptografi DES dan sebesar 128 bit biner atau 32 heksadesimal atau 16 karakter jika menggunakan algoritma AES.
3) *Plaintext* yang akan dienkripsikan.
4) Pada tahap penghitungan nilai *avalanche effect*, parameter yang perlu diinputkan adalah *ciphertext* yang akan dibandingkan.

3.2.2 Pembuatan *external key*

Untuk tipe percobaan pertama diperlukan sejumlah *external key* dan untuk tipe percobaan kedua hanya diperlukan sebuah *external key*. Dalam menghasilkan kunci eksternal yang berbeda, terlebih

dahulu dibuat sebuah kunci eksternal yang terdiri dari karakter-karakter (huruf atau angka) dalam skripsi ini digunakan inputan berupa biner. Kemudian dilakukan perubahan pada bit-bit kunci eksternal ini sebesar n > 1 melalui proses *keygen* pada perangkat lunak, dimana n adalah bilangan genap, agar diperoleh kunci eksternal yang lainnya. Hal ini bertujuan untuk melihat seberapa besarkah perubahan kecil yang diberikan pada kunci eksternal mempengaruhi *ciphertext* yang dihasilkan dari proses enkripsi.

3.2.3 Pembuatan *plaintext*

Ada dua tipe percobaan yang dilakukan dalam pengerjaan Skripsi ini, sebagaimana yang telah dijelaskan pada sub bab 3.1. Untuk tipe percobaan yang pertama hanya diperlukan sebuah *file plaintext*, sedangkan untuk tipe kedua memerlukan sejumlah *file plaintext. File plaintext* ini berisi teks biasa.

Untuk memperoleh sejumlah *plaintext* yang berbeda pada tipe uji coba kedua, maka dibuat *file Plaintext1* terlebih dahulu dan setelah itu dilakukan modifikasi terhadap bit-bit *file Plaintext1* tersebut agar diperoleh *file Plaintext2, file Plaintext3,* hingga *file PlaintextX*. Modifikasi yang dilakukan berupa mengubah nilai biner dari bit-bit *file Plaintext1* sebanyak $n > 1$, dimana n adalah bilangan genap, menjadi kebalikan dari nilai biner tersebut dan perubahan dimulai dari bit yang paling kanan, seperti yang telah dicontohkan pada sub bab 3.1.

3.2.4 Enkripsi *file plaintext*

File plaintext yang telah dibuat pada sub bab 3.2.3 dienkripsikan dengan menggunakan parameter yang ada pada sub bab 3.2.1 yakni algoritma yang digunakan untuk enkripsi, *external key* dan *file plaintext* yang akan dienkripsikan. Setiap *plaintext* dienkripsikan dengan menggunakan kedua algoritma kriptografi, DES dan AES. Hasil dari proses enkripsi berupa *ciphertext* yang disimpan untuk nantinya digunakan sebagai input pada tahap penghitungan *avalanche effect.*

3.2.5 Penghitungan nilai *avalanche effect*

Pada tahap penghitungan nilai *avalanche effect,* diperlukan input berupa dua *file ciphertext* yang akan dibandingkan bit-bitnya. Isi dari

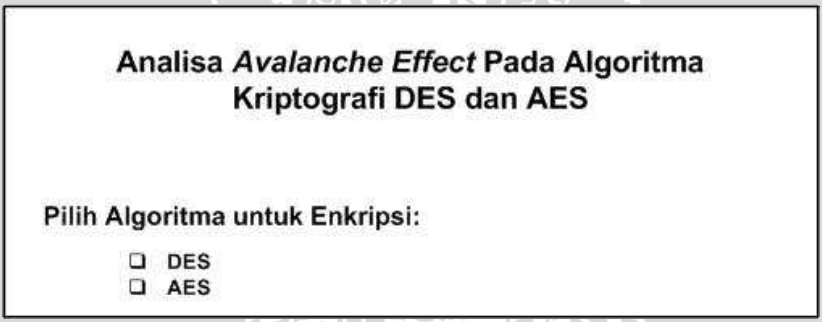
kedua *ciphertext* tersebut dikonversikan ke dalam nilai biner (0 dan 1), kemudian dilakukan perbandingan terhadap setiap bit yang berada pada posisi yang sama apakah berisi nilai yang sama atau tidak. Contoh penghitungan jumlah bit yang berbeda dapat dilihat pada sub bab 2.6. Setelah diperoleh jumlah bit yang berbeda dari kedua *file ciphertext* yang diinputkan, maka selanjutnya dihitung nilai *avalanche effect*-nya:

$$avalancheeffect = \frac{jumlah\,bit\,berbeda}{jumlah\,bit\,total\,Ciphertext A1} \times 100\%$$

atau

$$avalancheeffect = \frac{jumlah\,bit\,berbeda}{jumlah\,bit\,total\,Ciphertext 1P} \times 100\%$$

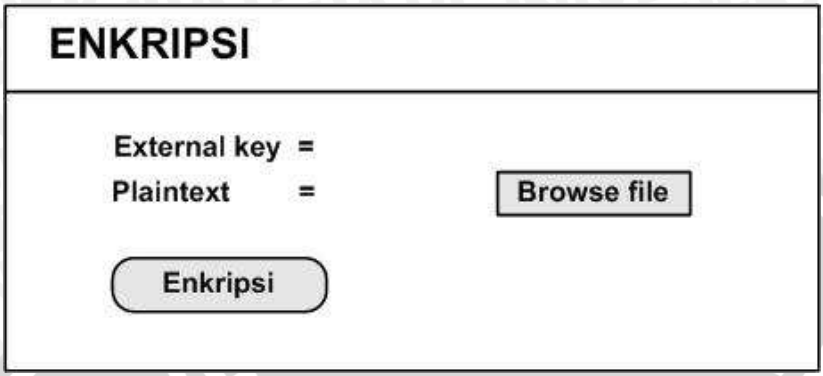
3.3 Perancangan Antarmuka



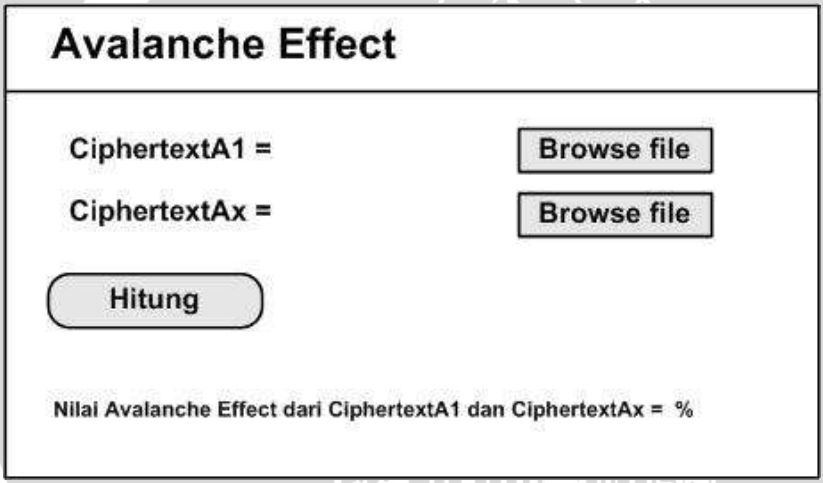
Gambar 3.4 Rancangan Antarmuka Halaman Utama

Gambar 3.4 adalah halaman utama hanya terdapat pilihan apakah user akan mengenkripsikan *plaintext* dengan menggunakan algoritma kriptografi DES atau AES.

Gambar 3.5 adalah halaman enkripsi, dilakukan proses enkripsi terhadap *plaintext* dengan menggunakan algoritma DES atau AES, tergantung pada input yang diberikan user dari halaman utama. Untuk melakukan proses enkripsi diperlukan parameter berupa *external key* dan *plaintext* sebagaimana yang telah dijelaskan pada sub bab 3.2.1.



Gambar 3.5 Rancangan Antarmuka Halaman Enkripsi



Gambar 3.6 Rancangan Antarmuka Halaman Penghitungan Nilai *Avalanche Effect*

Gambar 3.6 adalah halaman penghitungan nilai *avalanche effect* diperlukan inputan berupa *file CiphertextA1* dan *CiphertextAx* atau *Ciphertext1P* dan *CiphertextXP* yang akan dibandingkan dan dihitung nilai *avalanche effect*-nya. Output dari perangkat lunak ini adalah nilai *avalanche effect* dari *file ciphertext* yang dibandingkan.

3.4 Perancangan Uji Coba dan Evaluasi Hasil

Uji coba akan dilakukan terhadap perangkat lunak dengan menggunakan kedua jenis algoritma yakni algoritma kriptografi DES dan AES. Sebagaimana yang telah dijelaskan pada sub bab 3.1, akan dilakukan dua tipe uji coba pada pengerjaan Skripsi ini, yakni:

1) 1 *plaintext* diuji coba dengan menggunakan 20 kunci eksternal yang berbeda.
2) 20 *plaintext* yang berbeda diuji coba dengan menggunakan 1 kunci eksternal.

Kedua tipe percobaan tersebut dilakukan dengan tujuan untuk melihat seberapa besar perubahan kecil yang dilakukan pada *plaintext* atau *external key* akan memberikan pengaruhnya pada *ciphertext. Plaintext* yang digunakan untuk melakukan enkripsi dengan DES sama dengan *plaintext* yang digunakan untuk AES.

Setelah diperoleh nilai *avalanche effect* dari setiap tipe uji coba, maka dilakukan analisis terhadap hasil yang diberikan oleh setiap tipe percobaan baik dengan menggunakan algoritma DES maupun AES. Secara keseluruhan, terdapat 4 kali uji coba yakni:

1) tipe uji coba pertama dengan menggunakan DES
2) tipe uji coba kedua dengan menggunakan DES
3) tipe uji coba pertama dengan menggunakan AES
4) tipe uji coba kedua dengan menggunakan AES

Tabel 3.1 Rancangan tabel hasil uji untuk tipe uji coba pertama dan kedua

Pasangan *ciphertext*	**∑ bit *ciphertext* berbeda**	**∑ bit *ciphertextA1 atau ciphertext1P***	***Avalanche effect* (%)**

Keterangan Tabel 3.1 :

1) Pasangan *ciphertext* adalah *ciphertext* yang akan dihitung nilai *avalanche effect*-nya.
2) ∑ bit *ciphertext* berbeda adalah jumlah bit berbeda dari pasangan *ciphertext* yang dibandingkan satu per satu nilai bit-bitnya, yang terletak pada posisi yang sama.
3) ∑ bit *CiphertextA1* atau *Ciphertext1P* adalah jumlah keseluruhan bit yang dimiliki oleh *CiphertextA1* atau *Ciphertext1P*
4) $$Avalanche\ effect = \frac{jumlah\ bit\ berbeda}{jumlah\ bit\ total\ Ciphertext\ A1} \times 100\%$$

   Atau

   $$Avalanche\ effect = \frac{jumlah\ bit\ berbeda}{jumlah\ bit\ total\ Ciphertext\ 1P} \times 100\%$$

Tabel 3.2 Rancangan tabel evaluasi tipe uji coba pertama

Jumlah n External key DES	***Avalanche effect DES***	**Jumlah n External key AES**	***Avalanche effect AES***

Keterangan Tabel 3.2 :

1) Jumlah n *external key* DES adalah banyaknya bit *externalkeyX* untuk DES yang berbeda dari bit *external key1*.
2) *Avalanche effect* DES adalah nilai *avalanche effect* dari *ciphertextAX* jika dibandingkan dengan *ciphertextA1* yang dienkripsikan dengan menggunakan algoritma DES.
3) Jumlah n *external key* AES adalah banyaknya bit *external keyX* untuk AES yang berbeda dari bit *external key1*.
4) *Avalanche effect* AES adalah nilai *avalanche effect* dari *ciphertextAX* jika dibandingkan dengan *ciphertextA1* yang dienkripsikan dengan menggunakan algoritma AES.

Tabel 3.3 Rancangan tabel evaluasi tipe uji coba kedua

Jumlah n *Plaintext* DES	***Avalanche effect DES***	**Jumlah n *Plaintext* AES**	***Avalanche effect AES***

Keterangan Tabel 3.3 :

1) Jumlah n *plaintext* DES adalah banyaknya bit *plaintextX* untuk DES yang berbeda dari bit *Plaintext1*.
2) *Avalanche effect* DES adalah nilai *avalanche effect* dari *Ciphertext1P* jika dibandingkan dengan *CiphertextXP* yang dienkripsikan dengan menggunakan algoritma DES.
3) Jumlah n *external key* AES adalah banyaknya bit *PlaintextX* untuk AES yang berbeda dari bit *Plaintext1*.
4) *Avalanche effect* AES adalah nilai *avalanche effect* dari *Ciphertext1P* jika dibandingkan dengan *CiphertextXP* yang dienkripsikan dengan menggunakan algoritma AES.

3.5 Contoh Perhitungan Enkripsi DES

Plaintex : 0123456789ABCDEF (hexadesimal)
Key : 133457799BBCDFF1

Biner
Plaintext : 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Kemudian *plaintext* dibagi menjadi 2 bagian :

L (*left*) : 0000 0001 0010 0011 0100 0101 0110 0111
R (*right*) : 1000 1001 1010 1011 1100 1101 1110 1111

key : 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Membuat 16 *subkey*
Melakukan permutasi *key* (*eksternal key*) dengan menggunakan tabel 2.2 agar mengasilkan 56 bit *key*

Key setelah dilakukan permutasi : 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

56 bit key dibagi 2 menjadi bagian kiri dan kanan
C0 (kiri) : 1111000 0110011 0010101 0101111
D0 (kanan) : 0101010 1011001 1001111 0001111

Dengan C0 dan D0 yang telah diketahui, maka dapat dibuat 16 blok Cn dan Dn 1<=n<=16
Lakukan pergeseran bit ke kiri pada C0 dan D0, pergeseran bit sesuai dengan tabel 2.3 untuk mendapatkan C1 dan D1 sampai mendapatkan C16 dan D16.

Geser 1 bit ke kiri C0 dan D0 untuk mendapatkan C1 dan D1
C1 : 1110000 1100110 0101010 1011111
D1 : 1010101 0110011 0011110 0011110

Geser 1 bit ke kiri C1 dan D1 untuk mendapatkan C2 dan D2
C2 : 1100001 1001100 1010101 0111111
D2 : 0101010 1100110 0111100 0111101

Geser 2 bit ke kiri C2 dan D2 untuk mendapatkan C3 dan D3
C3 : 0000110 0110010 1010101 0111111
D3 : 0101011 0011001 1110001 1110101
.
.
Dilakukan langkah yang sama untuk mendapatkan C4 dan D4 sampai C15 dan D15
.
.
Geser 1 bit ke kiri C15 dan D15 untuk mendapatkan C16 dan D16
C16 : 1111000 0110011 0010101 0101111
D16 : 0101010 1011001 1001111 0001111

Untuk mendapatkan kunci pada setiap putaran *subkey* (Kn) maka dilakukan permutasi menggunakan tabel 2.4 terhadap Cn dan Dn.
C1 D1 : 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

K1 : 000110 110000 001011 101111 111111 000111 000001 110010

C2 D2 : 1100001 1001100 1010101 0111111 0101010 1100110 0111100 0111101
K2 : 011110 011010 111011 011001 110110 111100 100111 100101

C3 D3 : 0000110 0110010 1010101 1111111 0101011 0011001 1110001 1110101
K3 : 010101 011111 110010 001010 010000 101100 111110 011001

.
.
Dilakukan langkah yang sama untuk mendapatkan K4 sampai K15
.
.
C16 C16 : 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111
K16 : 110010 110011 110110 001011 000011 100001 011111 110101

Melakukan enkripsi terhadap tiap 64 bit blok data plaintext

Dilakukan permutasi awal (*initial permutation*) dengan menggunakan tabel 2.5.
Plaintext : 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Plaintext setelah dilakukan permutasi awal :
1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Dari *Plaintext* yang terbentuk, kemudian dibagi menjadi 2 bagian L (*left*) dan R (*right*).
L0 : 1100 1100 0000 0000 1100 1100 1111 1111
R0 : 1111 0000 1010 1010 1111 0000 1010 1010

L dan R diproses dalam 16 iterasi menggunakan fungsi F pada persamaan 2.6 dan 2.7 yang beroperasi dalam 2 blok, blok *Plaintext* 32 bit dan *subkey* Kn 48 bit, untuk menghasilkan sebuah blok yang berukuran 32 bit.

Round 1 (n=1)
R0 : 1111 0000 1010 1010 1111 0000 1010 1010
Lakukan fungsi ekpansi terhadap R0 yang berukuran 32 bit menjadi 48 bit dengan menggunakan tabel 2.6
E(R0) : 011110 100001 010101 010101 011110 100001 010101 010101

E(R0) ⊕ K1
E(R0) : 011110 100001 010101 010101 011110 100001 010101 010101
K1 : 000110 110000 001011 101111 111111 000111 000001 110010
———————————————————————— ⊕
011000 010001 011110 111010 100001 100110 010100 10111

Si adalah fungsi yang didefinisikan dalam tabel S-Box, Bi adalah blok 6 bit sebagai input input untuk S-Box.
B1 = 011000 = 24 ⟶ S1 = 5 = 0101
B2 = 011110 = 17 ⟶ S2 = 12 = 1100
B3 = 011000 = 30 ⟶ S3 = 8 = 1000
B4 = 011000 = 58 ⟶ S4 = 2 = 0010
B5 = 011000 = 33 ⟶ S5 = 11 = 1011
B6 = 011000 = 38 ⟶ S6 = 5 = 0101
B7 = 011000 = 20 ⟶ S7 = 9 = 1001
B8 = 011000 = 39 ⟶ S8 = 7 = 0111
Output : 0101 1100 1000 0010 1011 0101 1001 0111
Output dari S-Box menjadi input untuk proses permutasi dengan menggunakan tabel 2.8
Output setelah dilakukan permutasi : 0010 0011 0100 1010 1010 1001 1011 1011

Menghitung R1 :
L0 : 1100 1100 0000 0000 1100 1100 1111 1111
: 0010 0011 0100 1010 1010 1001 1011 1011

———————————————————— ⊕

R1 : 1110 1111 0100 1010 0110 0101 0100 0100
L1=R0 : 1111 0000 1010 1010 1111 0000 1010 1010

Round 2 (n=2)
R1 : 1110 1111 0100 1010 0110 0101 0100 0100
Lakukan fungsi ekpansi terhadap R1 yang berukuran 32 bit menjadi 48 bit dengan menggunakan tabel 2.6

E(R1) : 011101 011110 101001 010100 001100 001010 101000 001001

E(R1) ⊕ K2
E(R1) : 011101 011110 101001 010100 001100 001010 101000 001001
K2 : 011110 011010 111011 011001 110110 111100 100111 100101
———————————————————— ⊕
000011 000100 010010 001101 111010 110110 001111 101100

Si adalah fungsi yang didefinisikan dalam tabel S-Box, Bi adalah blok 6 bit sebagai input input untuk S-Box.
B1 = 000011 = 3 → S1 = 15 = 1111
B2 = 000100 = 4 → S2 = 8 = 1000
B3 = 010010 = 18 → S3 = 13 = 1101
B4 = 001101 = 13 → S4 = 0 = 0000
B5 = 111010 = 58 → S5 = 3 = 0011
B6 = 110110 = 54 → S6 = 10 = 1010
B7 = 001111 = 15 → S7 = 10 = 1010
B8 = 101100 = 44 → S8 = 14 = 1110
Output : 1111 1000 1101 0000 0011 1010 1010 1110
Output dari S-Box menjadi input untuk proses permutasi dengan menggunakan tabel 2.8
Output setelah dilakukan permutasi : 0011 1100 1010 1011 1000 0111 1010 0011

Menghitung R2 :

L1 : 1111 0000 1010 1010 1111 0000 1010 1010
: 0011 1100 1010 1011 1000 0111 1010 0011

——————————————————— ⊕

R2 : 1100 1100 0000 0001 0111 0111 0000 1001
L2=R1 : 1110 1111 0100 1010 0110 0101 0100 0100

.
.

Dilakukan langkah yang sama untuk mendapatkan R3 dan L3 sampai L15 dan L15

.

Round 16 (n=16)

R15 : 1100 0001 0011 1101 0111 0000 1111 1010

Lakukan fungsi ekpansi terhadap R15 yang berukuran 32 bit menjadi 48 bit dengan menggunakan tabel 2.6

E(R15) : 011000 000010 100111 111010 101110 100001 011111 110101

E(R15) ⊕ K16

E(R15) : 011000 000010 100111 111010 101110 100001 011111 110101

K16 : 110010 110011 110110 001011 000011 100001 011111 110101

——————————————————— ⊕

101010 110001 010001 110001 101101 000000 000000 000000

Si adalah fungsi yang didefinisikan dalam tabel S-Box, Bi adalah blok 6 bit sebagai input input untuk S-Box.

B1 = 101010 = 42 ⟶ S1 = 6 = 0110
B2 = 110001 = 49 ⟶ S2 = 11 = 1011
B3 = 010001 = 17 ⟶ S3 = 2 = 0010
B4 = 110001 = 49 ⟶ S4 = 9 = 1001
B5 = 101101 = 45 ⟶ S5 = 2 = 0010
B6 = 000000 = 0 ⟶ S6 = 12 = 1100
B7 = 000000 = 0 ⟶ S7 = 4 = 0100
B8 = 000000 = 0 ⟶ S8 = 13 = 1101

Output : 0110 1011 0010 1001 0010 1100 0100 1101

Output dari S-Box menjadi input untuk proses permutasi dengan menggunakan tabel 2.8
Output setelah dilakukan permutasi : 1101 1000 0001 1000 1100 1010 1110 1100

Menghitung R16 :
L15 : 0111 0101 1101 1101 0101 0100 0000 0010
: 1101 1000 0001 1000 1100 1010 1110 1100
——————————————————— ⊕
R16 : 1010 1101 1100 0101 1001 1110 1111 1110
L16=R15 : 1100 0001 0011 1101 0111 0000 1111 1010
R16 dan L16 digabung
R16 L16 : 1010 1101 1100 0101 1001 1110 1111 1110 1100 0001 0011 1101 0111 0000 1111 1010

Output dari R16 L16 kemudian di permutasi dengan menggunakan tabel 2.9, hasil dari permutasi adalah *chipertext*
Chipertext : 1111000 00000011 10111010 10110011 10010111 10110101 11001101 11010111

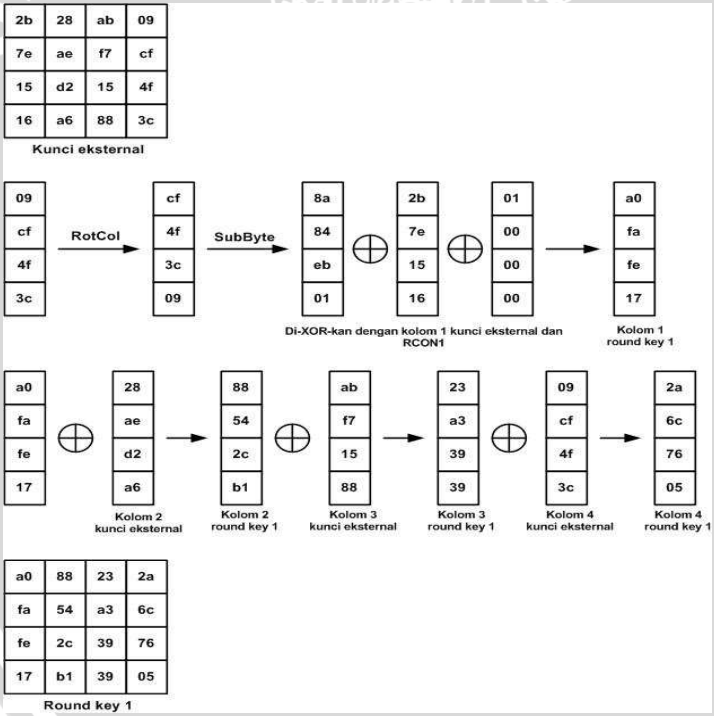
Chipertext : F003BAB397B5CDD7

3.6 Contoh Perhitungan Enkripsi AES

Plaintext = 3243f6a8885a308d313198a2e0370734 (hex)
External key = 2b7e151628aed2a6abf7158809cf4f3c (hex)

1) Proses inisialisasi kunci
Keterangan Gambar 3.7:
- ✓ User memberikan inputan pada perangkat lunak berupa *external key* (kunci eksternal) sebesar 16 karakter (128 bit). Kunci eksternal kemudian dikonversi ke dalam bentuk heksadesimalnya sebesar 32 heksadesimal dan dibagi menjadi 16 bagian, dan diisikan pada *state array* dimulai dari kotak kiri atas kemudian ke bawah.
- ✓ Untuk memperoleh *round key* 1 dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari kunci eksternal.

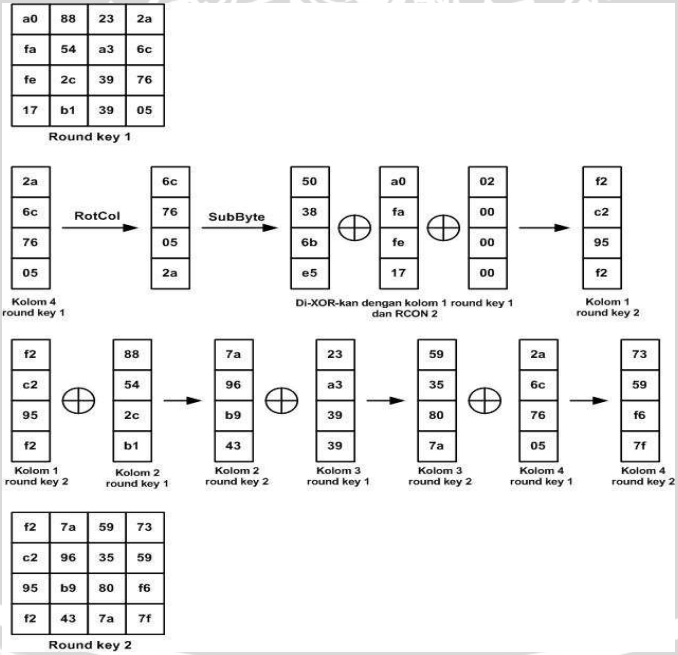
b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 kunci eksternal (kolom paling kiri) dan RCON1 sehingga menghasilkan kolom 1 dari *round key* 1.
d. Kolom 1 *round key* 1 di-XOR-kan dengan kolom 2 kunci eksternal sehingga menghasilkan kolom 2 *round key* 1.
e. Kolom 2 *round key* 1 di-XOR-kan dengan kolom 3 kunci eksternal sehingga menghasilkan kolom 3 *round key* 1.
f. Kolom 3 *round key* 1 di-XOR-kan dengan kolom 4 kunci eksternal sehingga menghasilkan kolom 4 *round key* 1.
g. Kolom 1 *round key* 1, kolom 2 *round key* 1, kolom 3 *round key* 1 dan kolom 4 *round key* 1 digabungkan menjadi satu dalam *state array* dan menghasilkan *round key* 1.



Gambar 3.7 Tahap memperoleh *round key* 1

Keterangan Gambar 3.8:

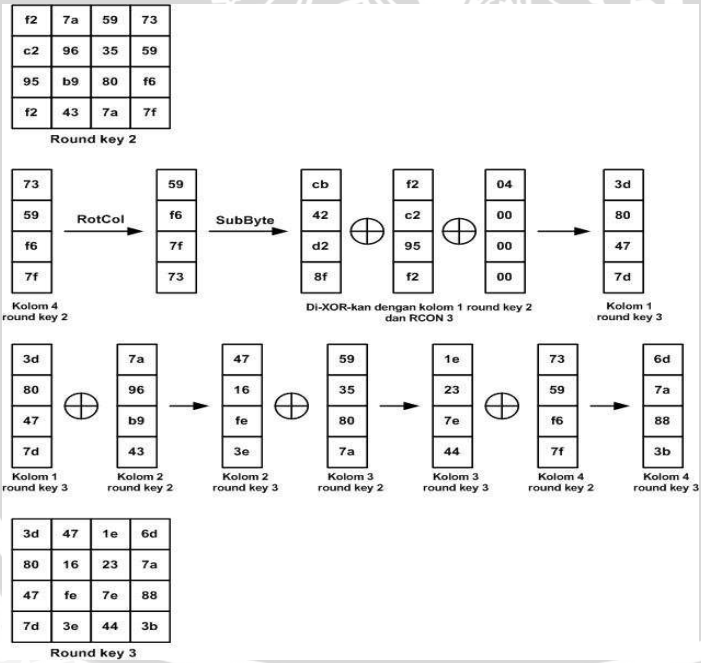
- ✓ Untuk memperoleh *round key* 2 dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key* 1.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key* 1 (kolom paling kiri) dan RCON2 sehingga menghasilkan kolom 1 dari *round key* 2.
  d. Kolom 1 *round key 2* di-XOR-kan dengan kolom 2 *round key* 1 sehingga menghasilkan kolom 2 *round key* 2.
  e. Kolom 2 *round key* 2 di-XOR-kan dengan kolom 3 *round key* 1 sehingga menghasilkan kolom 3 *round key* 2.
  f. Kolom 3 *round key* 2 di-XOR-kan dengan kolom 4 *round key* 1 sehingga menghasilkan kolom 4 *round key* 2.
  g. Kolom 1 *round key* 2, kolom 2 *round key* 2, kolom 3 *round key* 2 dan kolom 4 *round key* 2 digabungkan menjadi satu dalam *state array* dan menghasilkan *round key* 2.



Gambar 3.8 Tahap memperoleh *round key* 2

Keterangan Gambar 3.9:

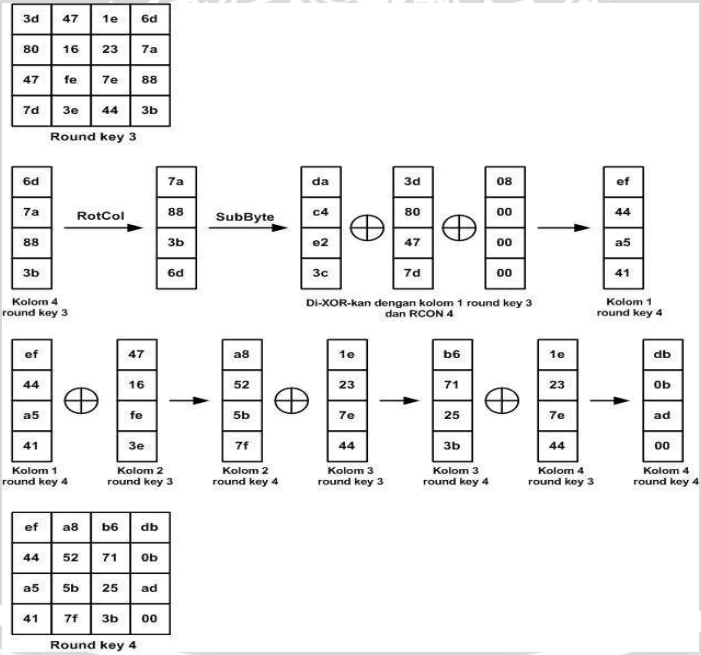
- ✓ Untuk memperoleh *round key* 3 dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 2*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key* 2 (kolom paling kiri) dan RCON3 sehingga menghasilkan kolom 1 dari *round key* 3.
  d. Kolom 1 *round key 3* di-XOR-kan dengan kolom 2 *round key 2* sehingga menghasilkan kolom 2 *round key 3*.
  e. Kolom 2 *round key 3* di-XOR-kan dengan kolom 3 *round key* 2 sehingga menghasilkan kolom 3 *round key 3*.
  f. Kolom 3 *round key 3* di-XOR-kan dengan kolom 4 *round key 2* sehingga menghasilkan kolom 4 *round key 3*.
  g. Kolom 1 *round key 3*, kolom 2 *round key 3*, kolom 3 *round key 3* dan kolom 4 *round key 3* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 3*.



Gambar 3.9 Tahap memperoleh *round key* 3

Keterangan Gambar 3.10:

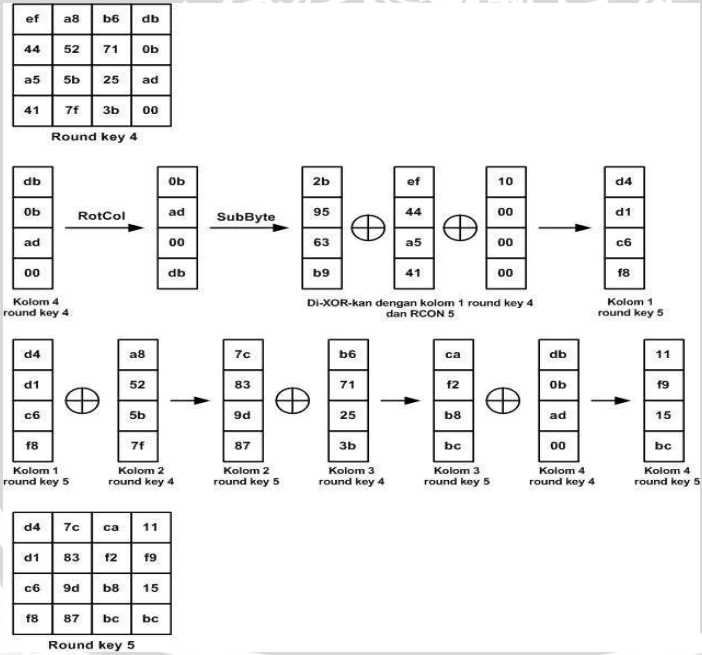
- ✓ Untuk memperoleh *round key 4* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 3*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 3* (kolom paling kiri) dan RCON4 sehingga menghasilkan kolom 1 dari *round key 4*.
  d. Kolom 1 *round key 4* di-XOR-kan dengan kolom 2 *round key 3* sehingga menghasilkan kolom 2 *round key 4*.
  e. Kolom 2 *round key 4* di-XOR-kan dengan kolom 3 *round key 3* sehingga menghasilkan kolom 3 *round key 4*.
  f. Kolom 3 *round key 4* di-XOR-kan dengan kolom 4 *round key 3* sehingga menghasilkan kolom 4 *round key 4*.
  g. Kolom 1 *round key 4*, kolom 2 *round key 4*, kolom 3 *round key 4* dan kolom 4 *round key 4* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 4*.



Gambar 3.10 Tahap memperoleh *round key 4*

Keterangan Gambar 3.11:

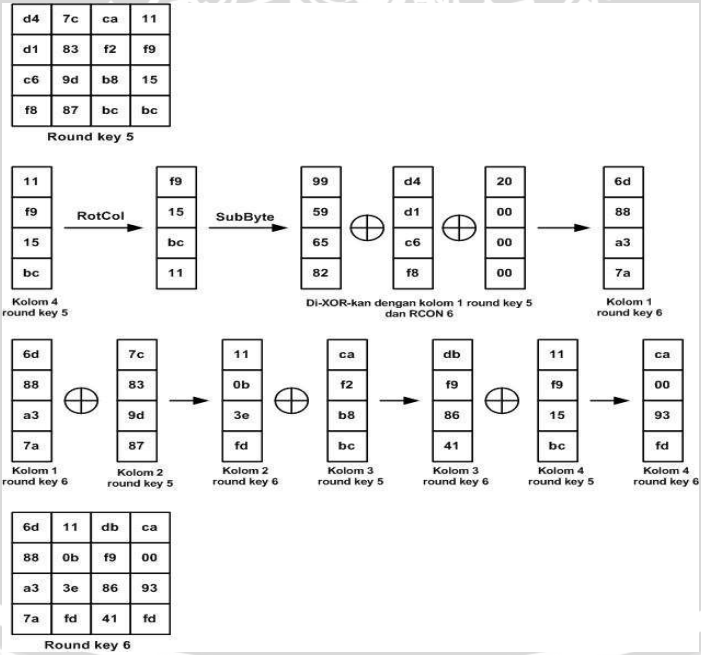
- ✓ Untuk memperoleh *round key 5* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 4*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 4* (kolom paling kiri) dan RCON5 sehingga menghasilkan kolom 1 dari *round key 5*.
  d. Kolom 1 *round key 5* di-XOR-kan dengan kolom 2 *round key 4* sehingga menghasilkan kolom 2 *round key 5*.
  e. Kolom 2 *round key 5* di-XOR-kan dengan kolom 3 *round key 4* sehingga menghasilkan kolom 3 *round key 5*.
  f. Kolom 3 *round key 5* di-XOR-kan dengan kolom 4 *round key 4* sehingga menghasilkan kolom 4 *round key 5*.
  g. Kolom 1 *round key 5*, kolom 2 *round key 5*, kolom 3 *round key 5* dan kolom 4 *round key 5* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 5*.



Gambar 3.11 Tahap memperoleh *round key 5*

Keterangan Gambar 3.12:

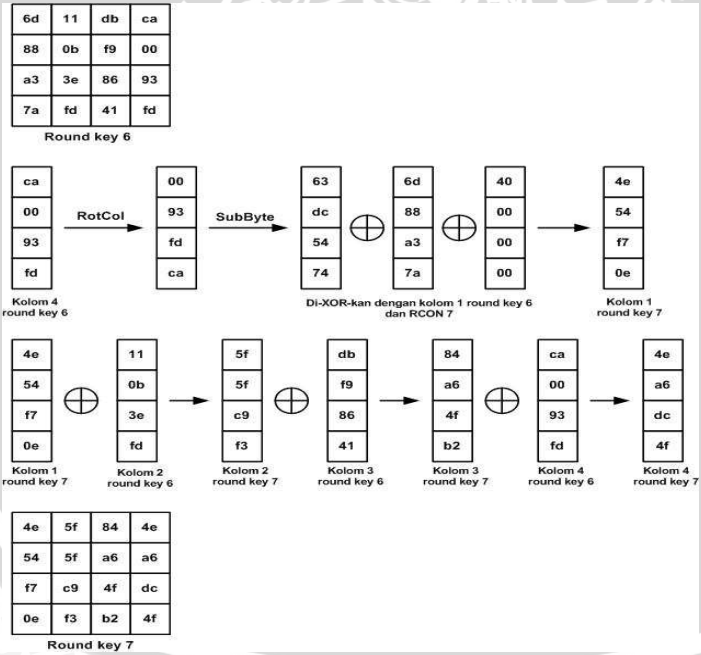
- ✓ Untuk memperoleh *round key 6* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key5*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 5* (kolom paling kiri) dan RCON6 sehingga menghasilkan kolom 1 dari *round key 6*.
  d. Kolom 1 *round key 6* di-XOR-kan dengan kolom 2 *round key 5* sehingga menghasilkan kolom 2 *round key 6*.
  e. Kolom 2 *round key 6* di-XOR-kan dengan kolom 3 *round key 5* sehingga menghasilkan kolom 3 *round key 6*.
  f. Kolom 3 *round key 6* di-XOR-kan dengan kolom 4 *round key 5* sehingga menghasilkan kolom 4 *round key 6*.
  g. Kolom 1 *round key 6*, kolom 2 *round key 6*, kolom 3 *round key 6* dan kolom 4 *round key 6* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 6*.



Gambar 3.12 Tahap memperoleh *round key 6*

Keterangan Gambar 3.13:

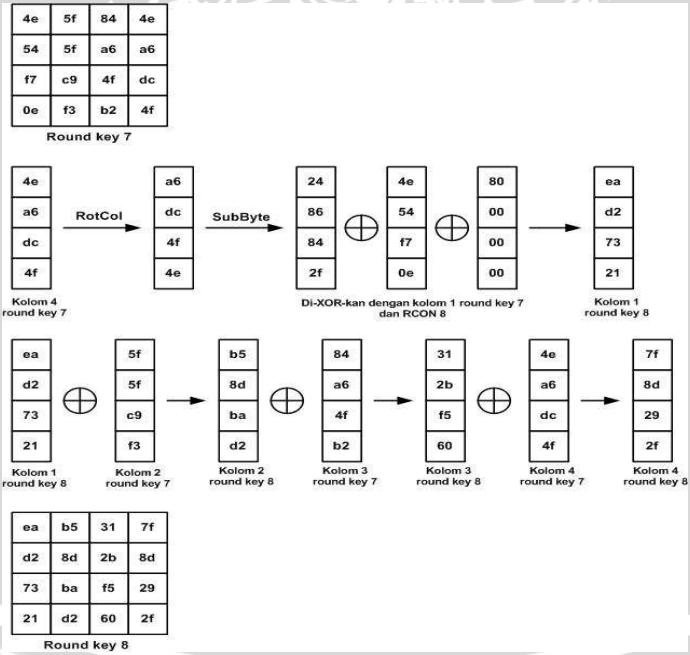
- ✓ Untuk memperoleh *round key 7* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 6*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 6* (kolom paling kiri) dan RCON7 sehingga menghasilkan kolom 1 dari *round key 7*.
  d. Kolom 1 *round key 7* di-XOR-kan dengan kolom 2 *round key 6* sehingga menghasilkan kolom 2 *round key 7*.
  e. Kolom 2 *round key 7* di-XOR-kan dengan kolom 3 *round key 6* sehingga menghasilkan kolom 3 *round key 7*.
  f. Kolom 3 *round key 7* di-XOR-kan dengan kolom 4 *round key 6* sehingga menghasilkan kolom 4 *round key 7*.
  g. Kolom 1 *round key 7*, kolom 2 *round key 7*, kolom 3 *round key 7* dan kolom 4 *round key 7* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 7*.



Gambar 3.13 Tahap memperoleh *round key 7*

Keterangan Gambar 3.14:

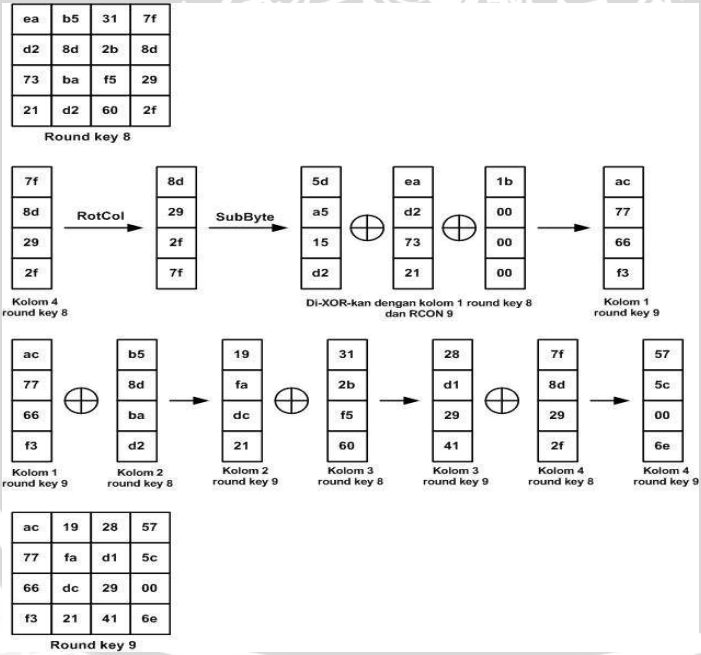
- ✓ Untuk memperoleh *round key 8* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 7*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 7* (kolom paling kiri) dan RCON8 sehingga menghasilkan kolom 1 dari *round key 8*.
  d. Kolom 1 *round key 8* di-XOR-kan dengan kolom 2 *round key 7* sehingga menghasilkan kolom 2 *round key 8*.
  e. Kolom 2 *round key 8* di-XOR-kan dengan kolom 3 *round key 7* sehingga menghasilkan kolom 3 *round key 8*.
  f. Kolom 3 *round key 8* di-XOR-kan dengan kolom 4 *round key 7* sehingga menghasilkan kolom 4 *round key 8*.
  g. Kolom 1 *round key 8*, kolom 2 *round key 8*, kolom 3 *round key 8* dan kolom 4 *round key 8* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 8*.



Gambar 3.14 Tahap memperoleh *round key 8*

Keterangan Gambar 3.15:

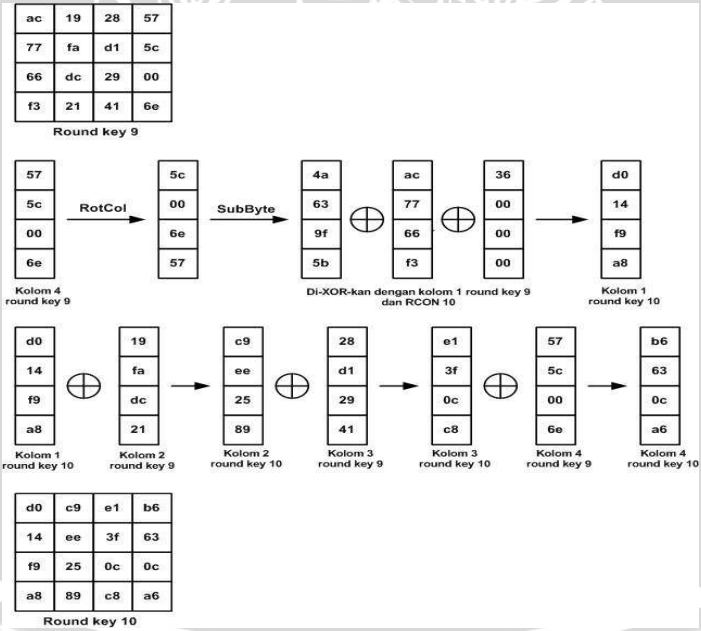
- ✓ Untuk memperoleh *round key 9* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 8*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 8* (kolom paling kiri) dan RCON9 sehingga menghasilkan kolom 1 dari *round key 9*.
  d. Kolom 1 *round key 9* di-XOR-kan dengan kolom 2 *round key 8* sehingga menghasilkan kolom 2 *round key 9*.
  e. Kolom 2 *round key 9* di-XOR-kan dengan kolom 3 *round key 8* sehingga menghasilkan kolom 3 *round key 9*.
  f. Kolom 3 *round key 9* di-XOR-kan dengan kolom 4 *round key 8* sehingga menghasilkan kolom 4 *round key 9*.
  g. Kolom 1 *round key 9*, kolom 2 *round key 9*, kolom 3 *round key 9* dan kolom 4 *round key 9* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 9*.



Gambar 3.15 Tahap memperoleh *round key 9*

Keterangan Gambar 3.16:

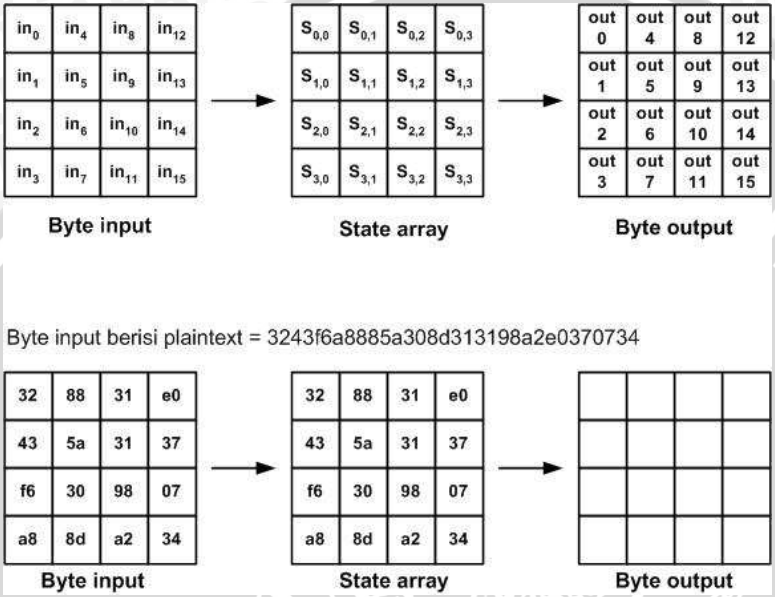
- ✓ Untuk memperoleh *round key 10* dilakukan tahap-tahap berikut:
  a. Transformasi RotCol terhadap kolom ke-4 (paling kanan) dari *round key 9*.
  b. Transformasi SubBytes terhadap kolom hasil dari RotCol tahap a.
  c. Kolom hasil transformasi SubBytes (tahap b) di-XOR-kan dengan kolom 1 *round key 9* (kolom paling kiri) dan RCON10 sehingga menghasilkan kolom 1 dari *round key 10*.
  d. Kolom 1 *round key 10* di-XOR-kan dengan kolom 2 *round key 9* sehingga menghasilkan kolom 2 *round key 10*.
  e. Kolom 2 *round key 10* di-XOR-kan dengan kolom 3 *round key 9* sehingga menghasilkan kolom 3 *round key 10*.
  f. Kolom 3 *round key 10* di-XOR-kan dengan kolom 4 *round key 9* sehingga menghasilkan kolom 4 *round key 10*.
  g. Kolom 1 *round key 10*, kolom 2 *round key 10*, kolom 3 *round key 10* dan kolom 4 *round key 10* digabungkan menjadi satu dalam *state array* dan menghasilkan *round key 10*.



Gambar 3.16 Tahap memperoleh *round key 10*

2) Proses enkripsi

Pada awal proses enkripsi, 16 byte data masukan, in_0, in_1, ..., in_{15}, disalin ke dalam *array state* (Gambar 3.17).

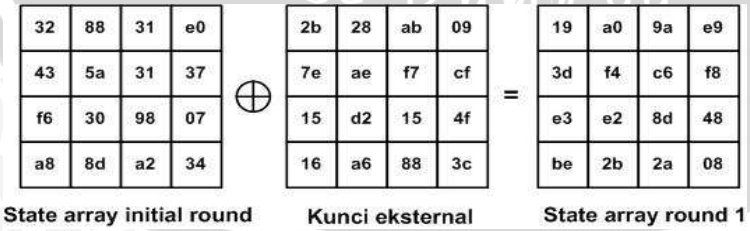


Gambar 3.17 *Byte input, state array* dan *byte output*

Byte input berisi *plaintext* sebesar 16 karakter (32 heksadesimal) yang kemudian isinya disalin ke *state array* untuk dilakukan proses enkripsi.

a. *Initial Round*

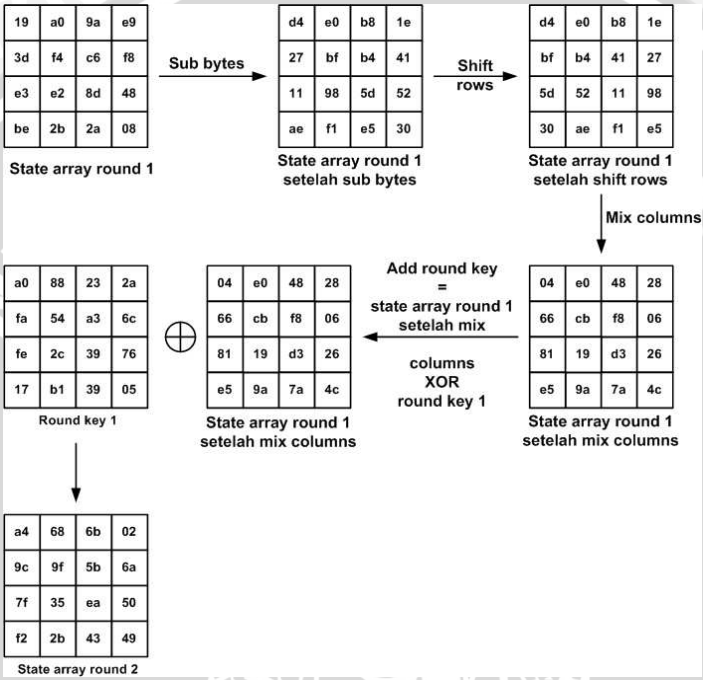
Initial round = state array $\oplus$ *external key*



Gambar 3.18 *Initial round*

Initial round adalah proses yang meng-XOR-kan *state array* dengan *external key* dan menghasilkan *state array round 1*(Gambar 3.18).

b. Round 1

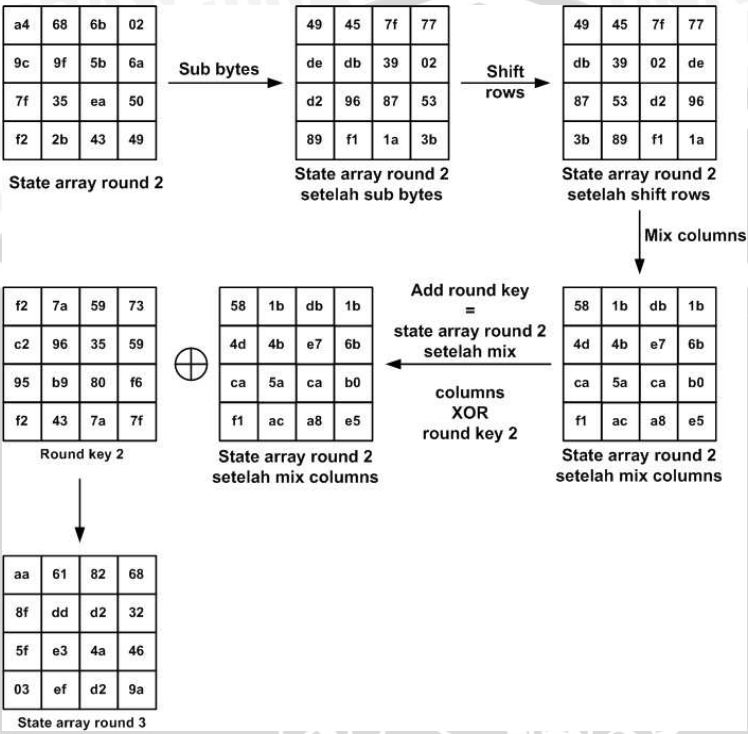


Gambar 3.19 Proses enkripsi *round 1*

Proses enkripsi *round 1* (Gambar 3.19) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 1*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 1* hasil proses SubBytes.
- ✓ Transformasi *MixColumns* terhadap *state array round 1* hasil proses *ShiftRows*.
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 1* hasil proses *MixColumns* dengan *round key 1*, yang menghasilkan *state array round 2*.

c. Round 2

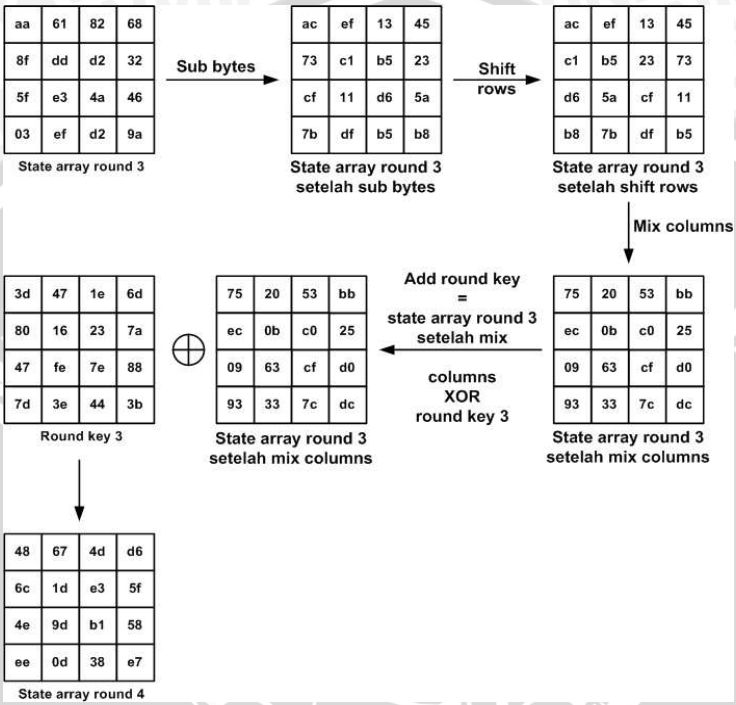


Gambar 3.20 Proses enkripsi *round 2*

Proses enkripsi *round 2* (Gambar 3.20) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 2*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 2* hasil proses SubBytes.
- ✓ Transformasi *MixColumns* terhadap *state array round 2* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 2* hasil proses *MixColumns* dengan *round key 2*, yang menghasilkan *state array round 3.*

d. Round 3

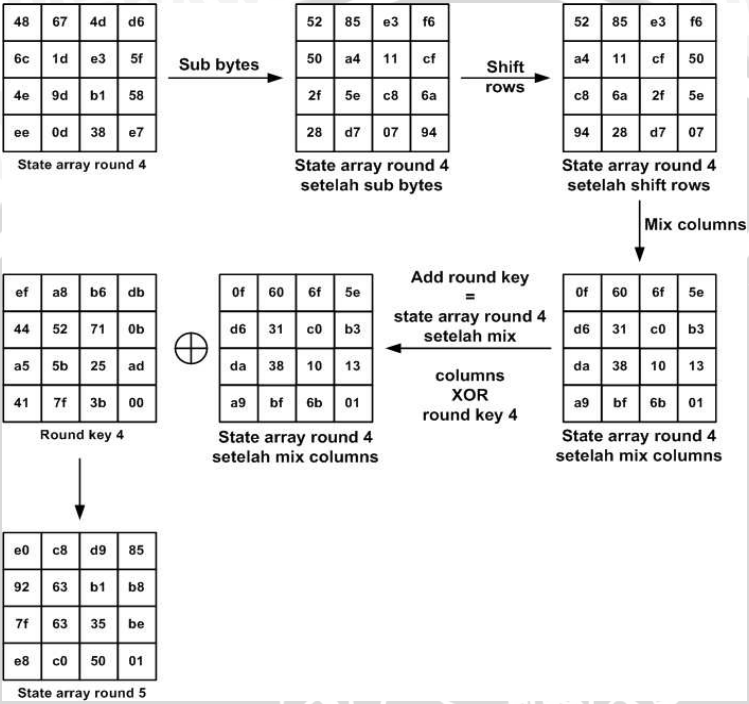


Gambar 3.21 Proses enkripsi *round 3*

Proses enkripsi *round 3* (Gambar 3.21) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 3*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 3* hasil proses SubBytes.
- ✓ Transformasi *MixColumns* terhadap *state array round 3* hasil proses *ShiftRows*.
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 3* hasil proses *MixColumns* dengan *round key 3*, yang menghasilkan *state array round 4.*

e. Round 4

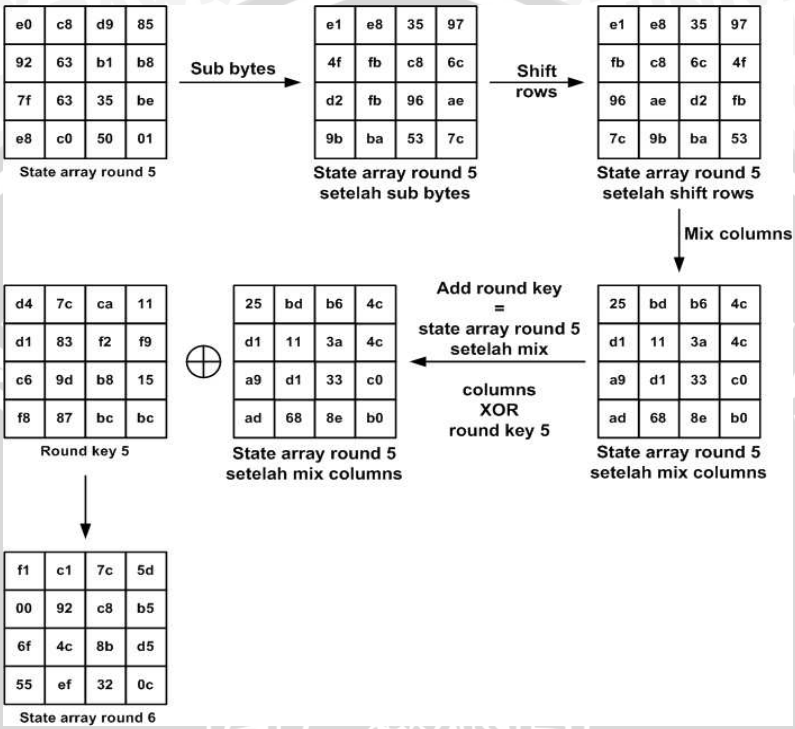


Gambar 3.22 Proses enkripsi *round 4*

Proses enkripsi *round 4* (Gambar 3.22)dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 4*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 4* hasil proses SubBytes.
- ✓ Transformasi *MixColumns* terhadap *state array round 4* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 4* hasil proses *MixColumns* dengan *round key 4*, yang menghasilkan *state array round 5*.

f. Round 5

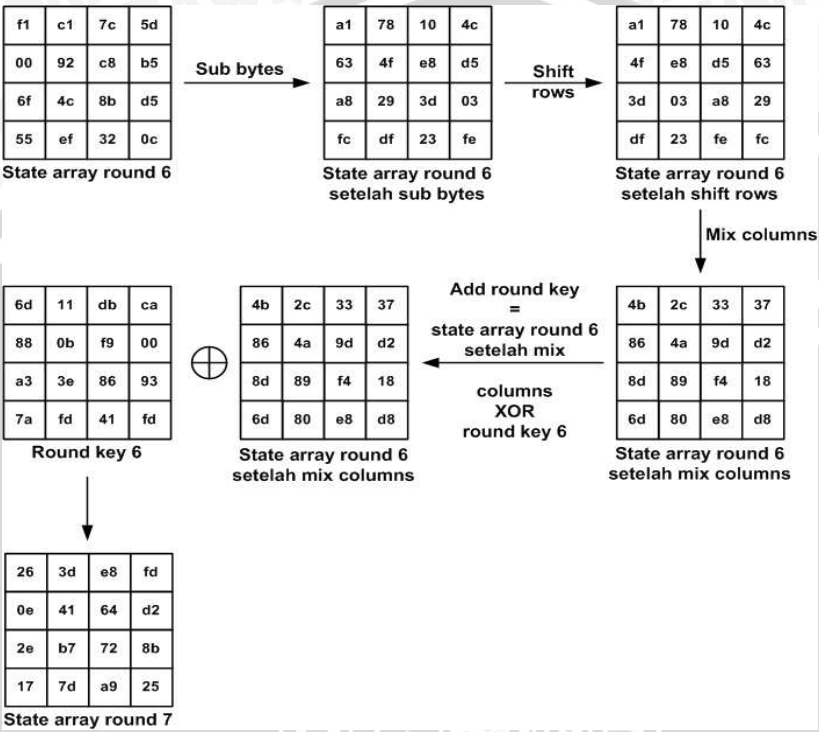


Gambar 3.23 Proses enkripsi *round 5*

Proses enkripsi *round 5* (Gambar 3.23) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 5*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 5* hasil proses *SubBytes.*
- ✓ Transformasi *MixColumns* terhadap *state array round 5* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 5* hasil proses *MixColumns* dengan *round key 5*, yang menghasilkan *state array round 6.*

g. Round 6

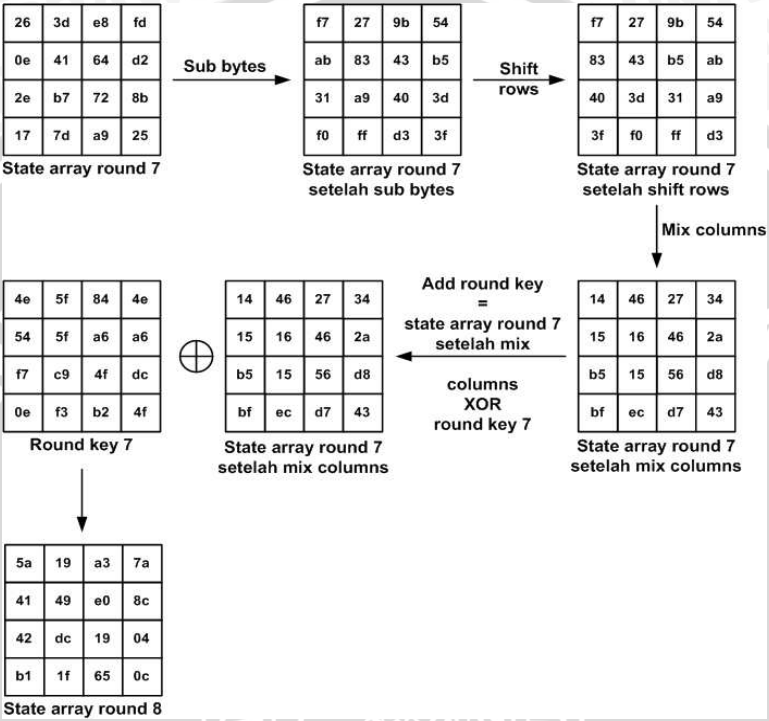


Gambar 3.24 Proses enkripsi *round 6*

Proses enkripsi *round 6* (Gambar 3.24) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 6.*
- ✓ Transformasi *ShiftRows* terhadap *state array round 6* hasil proses *SubBytes.*
- ✓ Transformasi *MixColumns* terhadap *state array round 6* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 6* hasil proses *MixColumns* dengan *round key 6*, yang menghasilkan *state array round 7.*

h. Round 7

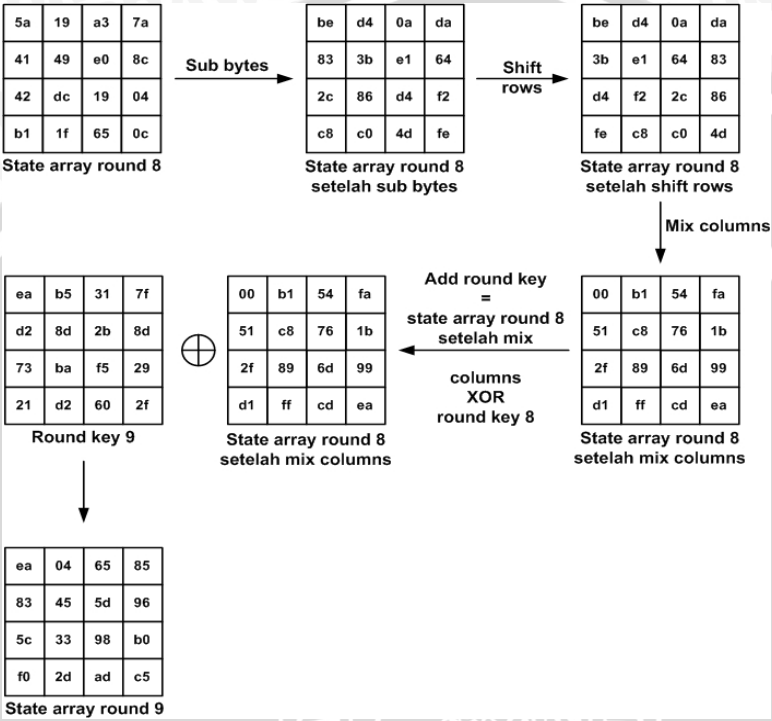


Gambar 3.25 Proses enkripsi *round 7*

Proses enkripsi *round 7* (Gambar 3.25) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 7*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 7* hasil proses *SubBytes.*
- ✓ Transformasi *MixColumns* terhadap *state array round 7* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 7* hasil proses *MixColumns* dengan *round key 7*, yang menghasilkan *state array round 8.*

i. Round 8

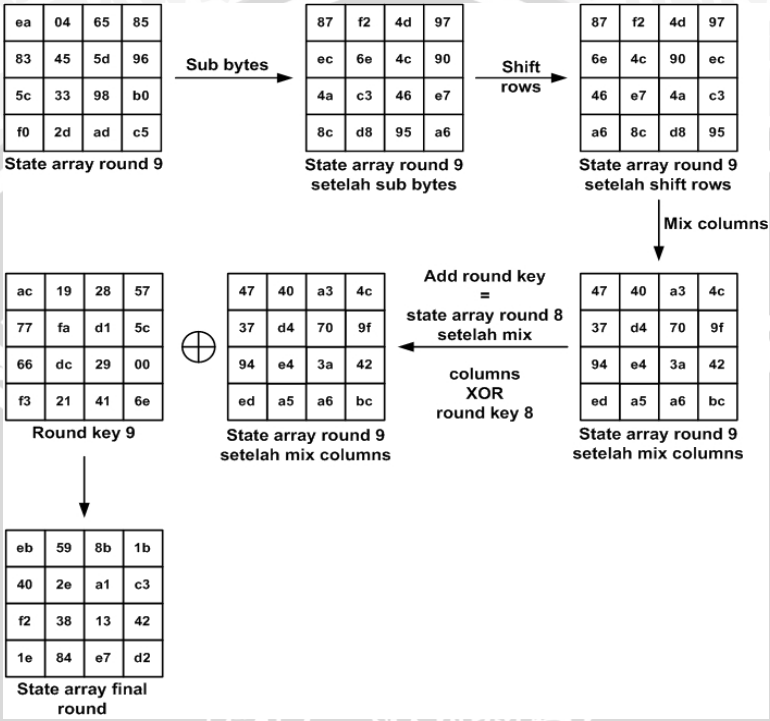


Gambar 3.26 Proses enkripsi *round 8*

Proses enkripsi *round 8* (Gambar 3.26) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 8*.
- ✓ Transformasi *ShiftRows* terhadap *state array round 8* hasil proses *SubBytes*.
- ✓ Transformasi *MixColumns* terhadap *state array round 8* hasil proses *ShiftRows*.
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array round 8* hasil proses *MixColumns* dengan *round key 8*, yang menghasilkan *state array final round*.

j. Round 9

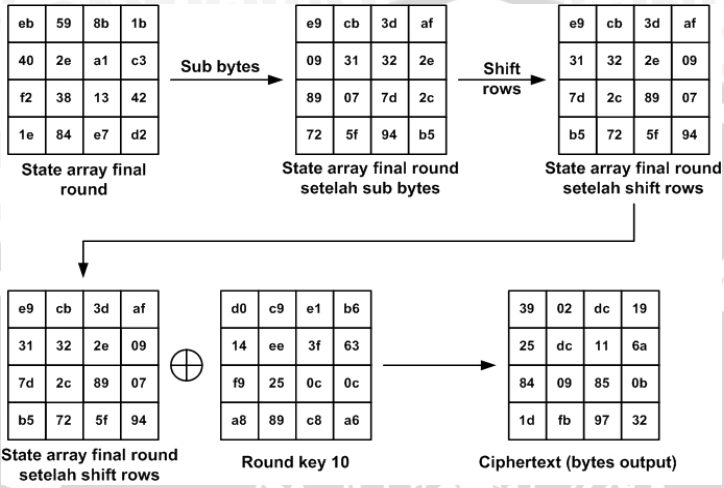


Gambar 3.27 Proses enkripsi *round 9*

Proses enkripsi *round 9* (Gambar 3.27) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array round 9.*
- ✓ Transformasi *ShiftRows* terhadap *state array round 9* hasil proses *SubBytes.*
- ✓ Transformasi *MixColumns* terhadap *state array round 9* hasil proses *ShiftRows.*
- ✓ Transformasi *AddRoundKey,* yakni meng-XOR-kan *state array round 9* hasil proses *MixColumns* dengan *round key 9,* yang menghasilkan *state array final round.*

k. Final round



Gambar 3.28 Proses enkripsi *final round*

Proses enkripsi *final round* (Gambar 3.28) dilakukan dalam beberapa tahap:

- ✓ Transformasi *SubBytes* terhadap *state array final round*.
- ✓ Transformasi *ShiftRows* terhadap *state array final round* hasil proses *SubBytes.*
- ✓ Transformasi *AddRoundKey*, yakni meng-XOR-kan *state array final round* hasil proses *ShiftRows* dengan *round key 10*, yang menghasilkan *ciphertext.*

3) Output

Output dari proses enkripsi dengan menggunakan algoritma kriptografi AES tersebut adalah *ciphertext* seperti yang ada pada Gambar 3.29:

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

Gambar 3.29 *Ciphertext* AES

Plaintext input :

3243f6a8885a308d313198a2e0370734 (hex)

External key :

2b7e151628aed2a6abf7158809cf4f3c (hex)

Ciphertext output :

3925841d02dc09fbdc118597196a0b32 (hex)

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Sebelum melakukan implementasi sistem, beberapa syarat harus disiapkan untuk memenuhi kebutuhan dari program yang akan implementasikan baik dari segi perangkat keras (*hardware*) maupun perangkat lunak (*software*) komputer.

4.1 Lingkungan Implementasi

Lingkungan implementasi meliputi lingkungan perangkat keras serta lingkungan perangkat lunak.

4.1.1 Lingkungan perangkat keras

Perangkat keras yang digunakan dalam pengembangan sistem pembuatan perangkat lunak *avalanche effect* adalah:

1. Prosesor Intel(R) Pentium(R) 4 – 2.26 GHz.
2. RAM 256 MB.
3. *Harddisk* dengan kapasitas 40 GB.
4. Monitor.
5. Keyboard.
6. Mouse.
7. Speaker.
8. Microphone.

4.1.2 Lingkungan perangkat lunak

Perangkat lunak yang digunakan dalam pengembangan sistem *avalanche effect* ini adalah :

1. Sistem Operasi *Microsoft Windows XP Professional Service Pack 2.*
2. *Microsoft Visual C# 2008 Express Edition.*

4.2 Implementasi Program

Berdasarkan perancangan perangkat lunak pada subbab 3.2 maka pada subbab ini akan dibahas mengenai implementasi dari perancangan tersebut.

4.2.1 Pembuatan *External Key*

Dalam uji coba perangkat lunak *avalanche effect* ini e*xternal key* untuk algoritma enkripsi DES adalah 0000000000000000000000000000000000000000000000000000000000000000 000000000. Dari *external key* tersebut kemudian dibentuk *external key* lainnya sebanyak 19 buah. Berikut adalah external key yang dipakai untuk algoritma enkripsi DES sebanyak 20 buah (Tabel 4.1).

Tabel 4.1 *External Key* untuk algoritma enkripsi DES

No	Biner
1	00000000000000000000000000000000000000000000000000000000000000000000000000000000
2	00000000000000000000000000000000000000000000000000000000000000000000000000000011
3	00000000000000000000000000000000000000000000000000000000000000000000000000001111
4	00000000000000000000000000000000000000000000000000000000000000000000000000111111
5	00000000000000000000000000000000000000000000000000000000000000000000000011111111
6	00000000000000000000000000000000000000000000000000000000000000000000000001111111111
7	00000000000000000000000000000000000000000000000000000000000000000000000111111111111
8	00000000000000000000000000000000000000000000000000000000000000000000001111111111111
9	00000000000000000000000000000000000000000000000000000000000000000000011111111111111
10	00000000000000000000000000000000000000000000000000000000000000000000111111111111111
11	00000000000000000000000000000000000000000000000000000000000000000001111111111111111
12	00000000000000000000000000000000000000000000000000000000000000000011111111111111111
13	00000000000000000000000000000000000000000000000000000000000000000111111111111111111
14	00000000000000000000000000000000000000000000000000000000000000001111111111111111111
15	00000000000000000000000000000000000000000000000000000000000000011111111111111111111
16	00000000000000000000000000000000000000000000000000000000000000111111111111111111111
17	00000000000000000000000000000000000000000000000000000000000001111111111111111111111
18	00000000000000000000000000000000000000000000000000000000000011111111111111111111111
19	00000000000000000000000000000000000000000000000000000000000111111111111111111111111
20	00000000000000000000000000000000000000000000000000000000001111111111111111111111111

Sedangkan *external key* untuk algoritma enkripsi AES adalah 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 000000000000000000. Dari *external key* tersebut kemudian dibentuk *external key* lainnya sebanyak 19 buah. Berikut adalah *external key* yang dipakai untuk algoritma enkripsi AES sebanyak 20 buah (Tabel 4.2).

Tabel 4.2 *External Key* untuk algoritma enkripsi AES

No	Biner
1	00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000

2	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000 00000011
3	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000 00001111
4	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000 00111111
5	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000 11111111
6	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000011 11111111
7	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000001111 11111111
8	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000111111 11111111
9	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000011111111 11111111
10	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000001111111111 11111111
11	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000111111111111 11111111
12	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000011111111111111 11111111
13	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000001111111111111111 11111111
14	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000111111111111111111 11111111
15	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000011111111111111111111 11111111
16	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000001111111111111111111111 11111111
17	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000111111111111111111111111 11111111
18	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000011111111111111111111111111 11111111
19	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000001111111111111111111111111111

	11111111
20	000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000111111111111111111111111111111 11111111

4.2.2 Pembuatan *Plaintext*

Dalam uji coba perangkat lunak *avalanche effect* ini *plaintext* yang dipakai adalah ”penyandian data dapat dilakukan dengan kriptografi” atau dalam bentuk biner :
1110000011001010110111001111001011000010110111001100100
0110100101100001011011100010000001100100011000010111010
0011000010010000001100100011000010111000000110000101110 1
0000100000011001000110100101101100011000010110101101110
1010110101101100001011011100010000001100100011001010110
1110011001110110000101101110001000000110101101110010011
0100101110000001110100011011110110011101110010011000010 1
1001100110100100001101000001010. Dari *plaintext* tersebut kemudian dibentuk 19 *plaintext* yang berbeda. Tabel 4.3 adalah *plaintext* yang dipakai dalam uji coba perangkat lunak *avalanche effect* ini.

Tabel 4.3 *Plaintext*

No	Biner
1	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000011000010111010000100000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 0110011101110010011000010110011001101001000011010000 1010
2	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000011000010111010000100000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010000110100001001
3	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000011000010111010000100000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 0110011101110010011000010110011001101001000011010000 0101
4	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001

	001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010000110100110101
5	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010000110111110101
6	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010000111011110101
7	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010000001011110101
8	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010010011001011110101
9	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010011111001011110101
10	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011010101111001011110101
11	011100000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000110010001100001011100000110000101110100001000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100110000101100110011001101111001011110101

12	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
13	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
14	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
15	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
16	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
17	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
18	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110
19	0111000001100101011011100111100101100001011011100110010001100101100010111001111001011000010110111001000000110010001100010101101110011001110110000101101110001000000110010001100101011011100110011101100001011011100000011010111011100100110100101110000011101000110111101100111000010110

	111000100000011010110111001001101001011100000111010001101111 01100111011100100110111010011001100101101111001011110101
20	0111000000110010101101110011110010110000101101110011001000110 100101100001011011100010000001100100011000010111010001100001 001000000011001000110000101110000001100001011101000010000000110 010001101001011011000110000101101011011101010110101101100001 011011100010000001100100011001010110111001100111011000010110 111000100000011010110111001001101001011100000111010001101111 01100111011100100101111010011001100101101111001011110101

4.2.3 Implementasi Enkripsi DES

Input dari proses enkripsi DES adalah *file* teks dan *external key* yang tersusun atas karakter (text), hexadesimal atau biner. Tiap karakter (text), hexadesimal atau biner tersebut mempunyai kode ascii yang unik. Nilai dari kode ascii tersebut yang kemudian diubah ke dalam bentuk biner dan dipakai dalam proses enkripsi. Dalam perangkat lunak *avalanche effect* ini, proses enkripsi DES dilakukan pada fungsi EncryptionStart (Gambar 4.1).

```
public override string EncryptionStart(string binaryText,
  string binarykey)
{
   #region Get 16 sub-keys using key
   string key_plus = this.DoPermutation(binarykey,
     DESData.pc_1);
   string C0 = "", D0 = "";
   C0 = this.SetLeftHalvesKey(key_plus);
   D0 = this.SetRightHalvesKey(key_plus);
   Keys keys = this.SetAllKeys(C0, D0);
   #endregion
   #region Encrypt process
   binaryText = this.setTextMutipleOf64Bits(binaryText);
   #region Initialize Progress Bar
   OnInitProgress(new
     Forms.ProgressInitArgs(binaryText.Length));
   #endregion
   StringBuilder EncryptedTextBuilder = new
     StringBuilder(binaryText.Length);
   //diambil tiap 64 bit
   for (int i = 0; i < (binaryText.Length / 64); i++)
   {
     //permutasi 64 bit
     string PermutatedText =
       this.DoPermutation(binaryText.Substring(i * 64, 64),
       DESData.ip);
     string L0 = "", R0 = "";
     L0 = this.SetLeftHalvesKey(PermutatedText);
     R0 = this.SetRightHalvesKey(PermutatedText);
     string FinalText = this.FinalEncription(L0, R0, keys,
       false);
```

```
      EncryptedTextBuilder.Append(FinalText);
      #region Increase Progress Bar
      OnIncrementProgress(new
        Forms.ProgressEventArgs(FinalText.Length));
      #endregion
    }
    return EncryptedTextBuilder.ToString();
    #endregion
  }
```

Gambar 4.1. *Source code* fungsi EncryptionStart

Dalam fungsi EncryptionStart, proses *permuted choice* 1 pada kunci eksternal untuk memperoleh 56 bit kunci internal dilakukan dengan pemanggilan fungsi DoPermutation (Gambar 4.2).

```
public string DoPermutation(string text, int[] order)
{
  StringBuilder PermutatedText = new
    StringBuilder(order.Length);
  for (int i = 0; i < order.Length; i++)
  {
    PermutatedText.Append(text[order[i] - 1]);
  }
  return PermutatedText.ToString();
}
```

Gambar 4.2. *Source code* fungsi DoPermutation

Setelah proses *permuted choice* 1 kemudian dilanjutkan dengan proses pembagian blok bit menjadi dua bagian melalui pemanggilan fungsi SetLeftHalvesKey dan SetRightHalvesKey. Sedangkan proses pergeseran bit terhadap kedua blok bit C0 dan D0 yang jumlahnya sesuai dengan tabel 2.2 dilakukan pada fungsi SetAllKeys (Gambar 4.3).

```
public Keys SetAllKeys(string C0, string D0)
{
  Keys keys = new Keys();
  keys.Cn[0] = C0;
  keys.Dn[0] = D0;
  for (int i = 1; i < keys.Cn.Length; i++)
  {
    keys.Cn[i] = this.LeftShift(keys.Cn[i - 1],
      DESData.nrOfShifts[i]);
    keys.Dn[i] = this.LeftShift(keys.Dn[i - 1],
      DESData.nrOfShifts[i]);
    keys.Kn[i - 1] = this.DoPermutation(keys.Cn[i] +
```

```
        keys.Dn[i], DESData.pc_2);
    }
    return keys;
}
```

Gambar 4.3. *Source code* fungsi SetAllKeys.

Kemudian dilanjutkan dengan proses menjadikan *plaintext* agar dapat dibagi kedalam 64 blok melalui pemanggilan fungsi setTextMutipleOf64Bits (Gambar 4.4).

```
public string setTextMutipleOf64Bits(string text)
{
    if ((text.Length % 64) != 0)
    {
        int maxLength = 0;
        maxLength = ((text.Length / 64) + 1) * 64;
        text = text.PadRight(maxLength, '0');
    }
    return text;
}
```

Gambar 4.4. *Source code* fungsi setTextMutipleOf64Bits.

Proses enkripsi dimulai dengan proses permutasi blok *plaintext* dengan matrik permutasi awal melalui pemanggilan fungsi DoPermutation. Proses permutasi dilakukan tiap 64 bit *plaintext*. Kemudian dilanjutkan dengan membagi hasil permutasi menjadi dua blok bit L0 dan R0. Kedua blok bit L0 dan R0 kemudian diproses menggunakan 16 putaran DES melalui fungsi FinalEncription (Gambar 4.5).

```
public string FinalEncription(string L0, string R0, Keys
    keys, bool IsReverse)
{
    string Ln = "", Rn = "", Ln_1 = L0, Rn_1 = R0;
    int i = 0;
    if (IsReverse == true)
    {
        i = 15;
    }
    while (this.IsEnough(i, IsReverse))
    {
        Ln = Rn_1;
        Rn = this.XOR(Ln_1, this.f(Rn_1, keys.Kn[i]));
        //Next Step of L1, R1 is L2 = R1, R2 = L1 + f(R1, K2),
hence, value of Step1's Ln, Rn is Rn_1, Ln_1 in Step2.
        Ln_1 = Ln;
```

```
    Rn_1 = Rn;
    if (IsReverse == false)
    {
      i += 1;
    }
    else
    {
      i -= 1;
    }
  }
  string R16L16 = Rn + Ln;
  string Encripted_Text = this.DoPermutation(R16L16,
    DESData.ip_1);
  return Encripted_Text;
}
```

Gambar 4.5. *Source code* fungsi FinalEncription.

4.2.4 Implementasi Enkripsi AES

Input dari proses enkripsi AES adalah *file* teks dan external key yang tersusun atas karakter (text), hexadesimal atau biner. Tiap karakter (text), hexadesimal atau biner tersebut kemudian diubah ke dalam bentuk biner dan dipakai dalam proses enkripsi. Dalam perangkat lunak *avalanche effect* ini, proses enkripsi AES dilakukan pada fungsi EncryptionStart (Gambar 4.6).

```
public override string EncryptionStart(string
  binarytext, string binarykey)
{
  StringBuilder binaryText = null;
  binaryText = new
    StringBuilder(BaseTransform.setTextMutipleOf128Bits(
      binarytext));
  StringBuilder EncryptedTextBuilder = new
    StringBuilder(binaryText.Length);
  #region Make All-round keys
  Matrix Matrix_CipherKey = new Matrix(binarykey);
  Keys key = new Keys();
  key.setCipherKey(Matrix_CipherKey);
  key = this.KeyExpansion(key, false);
  #endregion
  #region Initialize Progress Bar
  OnInitProgress(new
    Forms.ProgressInitArgs(binaryText.Length));
  #endregion
  for (int j = 0; j < (binaryText.Length / 128); j++)
  {
    Matrix state = new
      Matrix(binaryText.ToString().Substring(j * 128, 128));
    state = this.AddRoundKey(state, key, 0);
```

```
    for (int i = 1; i < 11; i++)
    {
      if (i == 10)
      {
        state = this.SubBytes(state, false);
        state = this.ShiftRows(state, false);
        state = this.AddRoundKey(state, key, i);
      }
      else
      {
        state = this.SubBytes(state, false);
        state = this.ShiftRows(state, false);
        state = this.MixColumns(state, false);
        state = this.AddRoundKey(state, key, i);
      }
    }
    EncryptedTextBuilder.Append(state.ToString());
    #region Increase Progress Bar
    OnIncrementProgress(new
      Forms.ProgressEventArgs(state.ToString().Length));
    #endregion
  }
  return EncryptedTextBuilder.ToString();
}
```

Gambar 4.6. *Source code* fungsi EncryptionStart.

Proses enkripsi AES dimulai dengan proses menjadikan *plaintext* agar dapat dibagi kedalam 128 blok melalui pemanggilan fungsi setTextMutipleOf128Bits (Gambar 4.7).

```
public static string setTextMutipleOf128Bits(string text)
{
  if ((text.Length % 128) != 0)
  {
    int maxLength = 0;
    maxLength = ((text.Length / 128) + 1) * 128;
    text = text.PadRight(maxLength, '0');
  }
  return text;
}
```

Gambar 4.7. *Source code* fungsi setTextMutipleOf128Bits.

Kemudian dilanjutkan dengan proses menjadikan kunci external kedalam bentuk matrik 4x4 dengan masing-masing elemen sebesar 1 byte (8 bit), melalui pemanggilan fungsi konstruktor Matrix. (Gambar 4.8).

```
public Matrix(string text)
            : this(text, 4, 4)
{}
public Matrix(string text, int rows, int columns)
{
  if (text.Length != columns * rows * 8)
  {
    text = text.PadRight(columns * rows * 8 - text.Length,
      '0');
  }
  matrix = new string[rows, columns];
  int count = 0;
  this.rows = rows;
  this.columns = columns;
  //susun binarykey menjadi matrik 4x4 mulai kolom pertama
kebawah,tiap elemen berisi 8 bit (1byte)
  for (int i = 0; i < columns; i++)
  {
    for (int j = 0; j < rows; j++)
    {
      matrix[j, i] = text.Substring(count * 8, 8);
      count++;
    }
  }
}
```

Gambar 4.8. *Source code* fungsi konstruktor Matrix..

Sebelum proses pembuatan 11 kunci internal, terlebih dahulu dilakukan inisialisasi matrix 4x4 untuk *round key*. Proses inisialisasi 10 buah *round key* dilakukan melalui pemanggilan fungsi setCipherKey (Gambar 4.9).

```
public void setCipherKey(Matrix CipherKey)
{
  if (RoundKeys.Count == 0)
  {
    this.RoundKeys.Add(CipherKey);
  }
  else
  {
    RoundKeys.Clear();
    RoundKeys.Add(CipherKey);
  }
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
  RoundKeys.Add(new Matrix(4, 4));
```

```
    RoundKeys.Add(new Matrix(4, 4));
    RoundKeys.Add(new Matrix(4, 4));
}
```

Gambar 4.9. *Source code* fungsi setCipherKey.

Dilanjutkan dengan proses pembuatan 11 kunci internal melalui pemanggilan fungsi KeyExpansion (Gambar 4.10).

```
public Keys KeyExpansion(Keys key, bool IsReverse)
{
  for (int i = 4; i < key.RoundKeys.Count * 4; i++)
  {
    string[] Wi_1 = key.RoundKeys[(i - 1) / 4].getWord((i -
      1) % 4);
    Matrix mat_Wi_1 = new Matrix(4, 1);
    mat_Wi_1.setWord(Wi_1, 0);
    if (i % 4 == 0)
    {
      Wi_1 = this.RotWord(Wi_1);
      mat_Wi_1.setWord(Wi_1, 0);
      mat_Wi_1 = this.SubBytes(mat_Wi_1, false);
      mat_Wi_1 = MatrixMultiplication.XOR(mat_Wi_1,
      TransformTabels.Rcon[(i - 1) / 4]);
    }
    Matrix Wi_4 = new Matrix(4, 1);
    Wi_4.setWord(key.RoundKeys[(i - 4) / 4].getWord((i - 4)
      % 4), 0);
    Matrix temp = MatrixMultiplication.XOR(mat_Wi_1, Wi_4);
    string[] Wi = temp.getWord(0);
    key.RoundKeys[i / 4].setWord(Wi, i % 4);
  }
  return key;
}
```

Gambar 4.10. *Source code* fungsi KeyExpansion.

Setelah mendapatkan 11 kunci external kemudian dilanjutkan proses enkripsi. Proses enkripsi kriptografi AES dilakukan tiap 128 bit *plaintext* dan dilakukan sebanyak 11 putaran yang pada setiap putarannya dilakukan operasi *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*.

4.2.5 Implementasi Dekripsi DES

Input dari proses dekripsi DES adalah *file chipertext* yang merupakan hasil enkripsi dari *file* teks dan external key yang tersusun atas karakter (text), hexadesimal atau biner. Tiap karakter

(text), hexadesimal atau biner tersebut kemudian diubah ke dalam bentuk biner dan dipakai dalam proses dekripsi. Dalam perangkat lunak *avalanche effect* ini, proses dekripsi DES dilakukan dengan fungsi pada Gambar 4.11

```
public override string DecryptionStart(string binarytext,
string binarykey)
{
            #region Get 16 sub-keys using key
            string key_plus = this.DoPermutation(binarykey,
DESData.pc_1);
            string C0 = "", D0 = "";
            C0 = this.SetLeftHalvesKey(key_plus);
            D0 = this.SetRightHalvesKey(key_plus);
            Keys keys = this.SetAllKeys(C0, D0);
            #endregion
            #region Decrypt process
            binarytext =
this.setTextMutipleOf64Bits(binarytext);

            #region Initialize Progress Bar
            OnInitProgress(new
Forms.ProgressInitArgs(binarytext.Length));
            #endregion
            StringBuilder DecryptedTextBuilder = new
StringBuilder(binarytext.Length);
            for (int i = 0; i < (binarytext.Length / 64);
i++)
            {
                string PermutatedText =
this.DoPermutation(binarytext.Substring(i * 64, 64),
DESData.ip);
                string L0 = "", R0 = "";
                L0 = this.SetLeftHalvesKey(PermutatedText);
                R0 = this.SetRightHalvesKey(PermutatedText);
                string FinalText = this.FinalEncription(L0,
R0, keys, true);
                #region It's for correct subtracted '0' that
have added for set text multiple of 64bit
                if ((i * 64 + 64) == binarytext.Length)
                {
                    StringBuilder last_text = new
StringBuilder(FinalText.TrimEnd('0'));
                    int count = FinalText.Length -
last_text.Length;
                    if ((count % 8) != 0)
                    {
                        count = 8 - (count % 8);
                    }
                    string append_text = "";
                    for (int k = 0; k < count; k++)
                    {
                        append_text += "0";
```

```
                    }
DecryptedTextBuilder.Append(last_text.ToString() +
append_text);
                }
                #endregion
                else
                {
                    DecryptedTextBuilder.Append(FinalText);
                }
                #region Increase Progress Bar
                OnIncrementProgress(new
Forms.ProgressEventArgs(FinalText.Length));
                #endregion
            }
            return DecryptedTextBuilder.ToString();
            #endregion
}
```

Gambar 4.11. *Source code* Dekripsi DES.

4.2.6 Implementasi Dekripsi AES

Input dari proses dekripsi AES adalah *file chipertext* yang merupakan hasil enkripsi dari *file* teks dan external key yang tersusun atas karakter (text), hexadesimal atau biner. Tiap karakter (text), hexadesimal atau biner tersebut kemudian diubah ke dalam bentuk biner dan dipakai dalam proses dekripsi. Dalam perangkat lunak *avalanche effect* ini, proses dekripsi AES dilakukan dengan fungsi pada Gambar 4.12

```
#region Decryption Process
        public override string DecryptionStart(string
binarytext, string binarykey)
        {

            StringBuilder DecryptedTextBuilder = new
StringBuilder(binarytext.Length);
            #region Make All-round keys
            Matrix Matrix_CipherKey = new Matrix(binarykey);
            Keys key = new Keys();
            key.setCipherKey(Matrix_CipherKey);
            key = this.KeyExpansion(key, false);
            #endregion
            #region Initialize Progress Bar
            OnInitProgress(new
Forms.ProgressInitArgs(binarytext.Length));
            #endregion
            for (int j = 0; j < (binarytext.Length / 128);
j++)
```

```
            {
                Matrix state = new
Matrix(binarytext.Substring(j * 128, 128));
state = this.AddRoundKey(state, key, 10);
                for (int i = 9; i >= 0; i--)
                {
                    if (i == 0)
                    {
                        state = this.ShiftRows(state, true);
                        state = this.SubBytes(state, true);
                        state = this.AddRoundKey(state, key, i);
                    }
                    else
                    {
                        state = this.ShiftRows(state, true);
                        state = this.SubBytes(state, true);
                        state = this.AddRoundKey(state, key, i);
                        state = this.MixColumns(state, true);
                    }
                }
                #region
                if ((j * 128 + 128) == binarytext.Length)
                {
                    StringBuilder last_text = new
StringBuilder(state.ToString().TrimEnd('0'));
                    int count = state.ToString().Length -
last_text.Length;
                    if ((count % 8) != 0)
                    {
                        count = 8 - (count % 8);
                    }
                    string append_text = "";
                    for (int k = 0; k < count; k++)
                    {
                        append_text += "0";
                    }
DecryptedTextBuilder.Append(last_text.ToString() +
append_text);
                }
                #endregion
                else
                {
DecryptedTextBuilder.Append(state.ToString());
                }
                #region Increase Progress Bar
                OnIncrementProgress(new
Forms.ProgressEventArgs(state.ToString().Length));
                #endregion
            }
            return DecryptedTextBuilder.ToString();
        }
        #endregion
```

Gambar 4.12. *Source code* Dekripsi AES.

4.2.7 Implementasi Perhitungan Avalanche Effect

Input dari proses perhitungan *Avalanche effect* adalah dua file ciphertext hasil enkripsi dari DES atau AES yang akan dibandingkan bit-bitnya. Isi dari kedua ciphertext tersebut dikonversikan ke dalam nilai biner (0 dan 1), kemudian dilakukan perbandingan terhadap setiap bit yang berada pada posisi yang sama apakah berisi nilai yang sama atau tidak, proses perhitungan *Avalanche effect* dilakukan dengan fungsi pada Gambar 4.13

```
class AvalancheEffect
    {
        public int totalbits=0;
        public int diffbits = 0;
        public double GetPercAvalancheEffect(string
ciphertext1, string ciphertext2)
        {
            int len1 = ciphertext1.Length;
            int len2 = ciphertext2.Length;
            if (len1 != len2)
                return -1;
            diffbits = 0;
            totalbits = ciphertext1.Length;
            for (int i = 0; i < totalbits; i++)
            {
                if (ciphertext1[i] != ciphertext2[i])
                {
                    diffbits++;
                }
            }
            return (diffbits * 100 / totalbits);
        }
    }
```

Gambar 4.13. *Source code Avalanche effect*.

4.2.8 Implementasi Keygen

Input dari proses *keygen* adalah sebuah kunci eksternal awal yang terdiri dari karakter-karakter (huruf atau angka) dalam skripsi ini digunakan inputan berupa biner. Kemudian dilakukan perubahan pada bit-bit kunci eksternal ini sebesar n > 1 melalui proses *keygen* pada perangkat lunak, dimana n adalah bilangan genap, agar diperoleh kunci eksternal yang lainnya, proses *keygen* dilakukan dengan fungsi pada Gambar 4.14

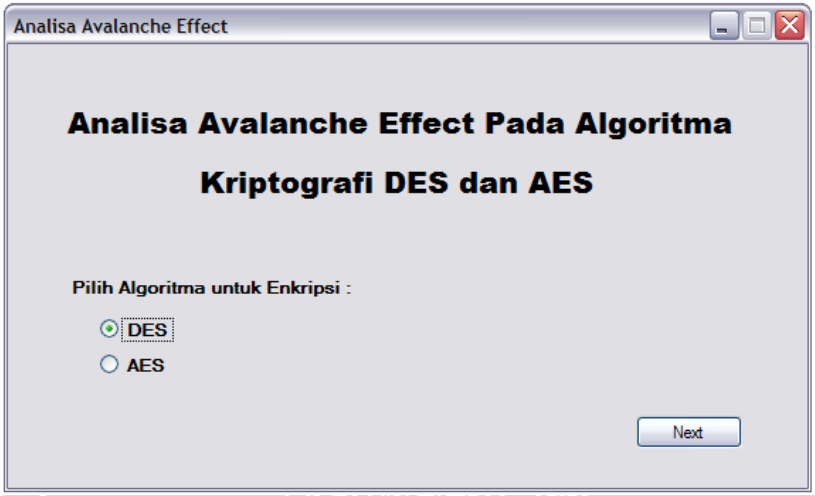
```
    private string inverse(string bin)
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < bin.Length; i++)
            {
                if (bin[i] == '0')
                    sb.Append('1');
                else if (bin[i] == '1')
                    sb.Append('0');
            }
            return sb.ToString();
        }
        private void btnGenerator_Click(object sender,
EventArgs e)
        {
            txtKeys.Clear();
            string extkey = txtExtKey.Text;
            extkey = extkey.ToUpper();
            if (rbTeks.Checked)
                extkey =
AES.BaseTransform.FromTextToBinary(extkey);
            if (rbHex.Checked)
                extkey =
AES.BaseTransform.FromHexToBinary(extkey);
            string oribin = extkey;
            int len = oribin.Length;
            txtKeys.Text = oribin + "\r\n";
            for (int i = len-2; i > 0; i-=2)
            {
                string newbin = inverse(oribin.Substring(i));
                string headbin = oribin.Substring(0,i);
                txtKeys.Text += headbin + newbin + "\r\n";
            }
        }
```

Gambar 4.14. *Source code* fungsi Keygen.

4.3 Implementasi Antarmuka

Berdasarkan rancangan antarmuka pada sub bab 3.3 maka dihasilkan antarmuka yang ditunjukkan pada Gambar 4.15, Gambar 4.16, Gambar 4.17 dan Gambar 4.18.

Dalam antarmuka *form* utama (Gambar 4.15) hanya terdapat dua pilihan yaitu algoritma enkripsi DES atau AES. *User* harus memilih salah satu algoritma tersebut dan menekan tombol Next untuk menuju *form* Enkripsi. Algoritma yang terpilih akan digunakan dalam proses enkripsi atau dekripsi pada *form* Enkripsi.



Gambar 4.15 Antarmuka *form* utama

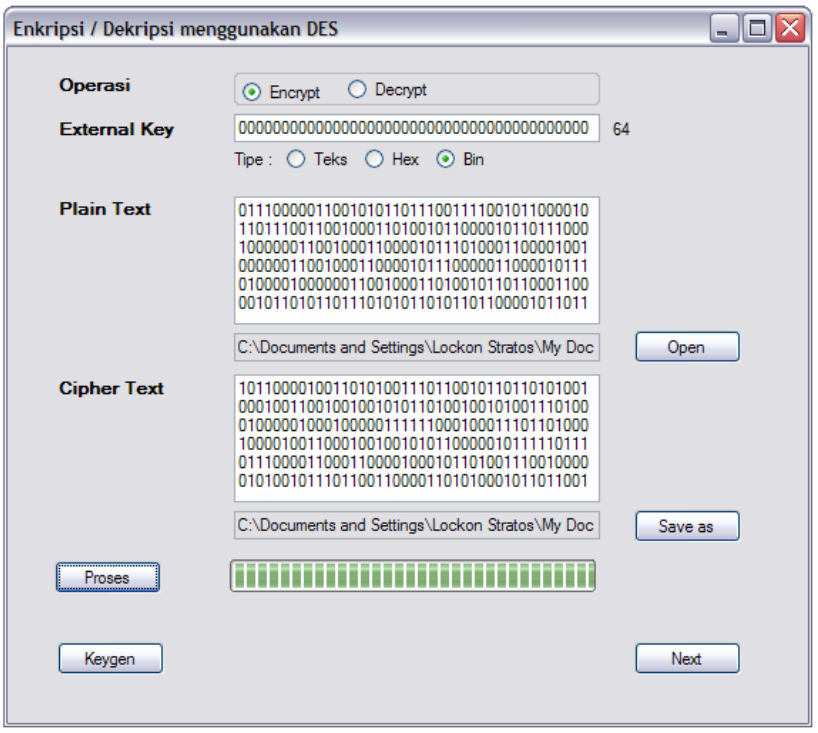
Dalam antarmuka *form* enkripsi dan dekripsi (Gambar 4.16) terdapat dua pilihan operasi yang akan dilakukan yaitu enkripsi atau dekripsi. Untuk melakukan proses enkripsi, dimulai dengan memilih *optionbox* Encrypt dilanjutkan dengan memasukkan *external key*. Input untuk *external key* bisa berupa teks, hexadesimal atau biner. Untuk melakukan proses dekripsi, dimulai dengan memilih *optionbox* Decrypt dilanjutkan dengan langkah yang sama dengan enkripsi.

Panjang *input* untuk *external key* berbeda-beda sesuai dengan jenis algoritma enkripsi dan tipe datanya. Untuk algoritma enkripsi DES panjang *input external key* yang diperbolehkan adalah 8 karakter untuk teks, 16 karakter untuk hexadesimal dan 64 karakter untuk biner. Sedangkan untuk algoritma enkripsi AES panjang *input external key* yang diperbolehkan adalah 16 karakter untuk teks, 32 karakter untuk hexadesimal dan 128 karakter untuk biner.

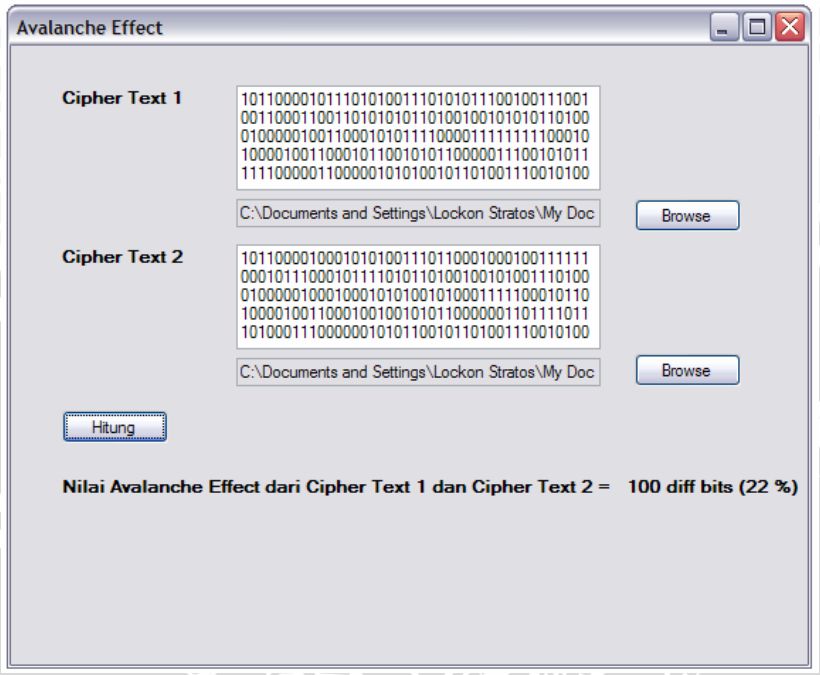
Setelah pemasukan *external key* kemudian dilanjutkan dengan pemasukan *plaintext* yang akan dienkripsi. Proses pemasukkan *plaintext* dapat dilakukan dengan dua cara yaitu dengan menekan tombol Browse lalu memilih *file* teks yang akan dienkripsi atau dengan cara memasukkan teks dalam bentuk binernya ke dalam *textbox*.

Setelah pemasukan *plaintext* kemudian dilanjutkan dengan proses penentuan *output file* hasil enkripsi. Penentuan *file output* enkripsi dilakukan dengan cara menekan tombol Browse lalu memilih lokasi dan mengetik nama *file output*.

Setelah memasukkan *external key*, *plaintext* dan lokasi *file output* kemudian dilanjutkan dengan menekan tombol Proses untuk melakukan proses enkripsi. Setelah proses enkripsi selesai akan muncul pesan yang menjelaskan bahwa proses enkripsi telah selesai.



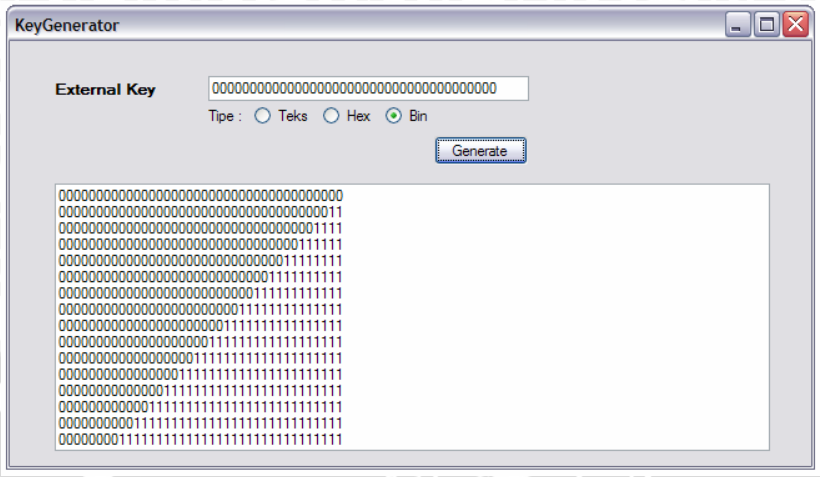
Gambar 4.16 Antarmuka *form* enkripsi



Gambar 4.17 Antarmuka *form* Penghitungan Nilai *Avalanche Effect*

Dalam antarmuka *form* Penghitungan Nilai *Avalanche Effect* (Gambar 4.17) terdapat dua *input* yaitu *ciphertext* 1 dan *ciphertext* 2. *User* dapat memasukkan *ciphertext* dengan cara menekan tombol Browse kemudian memilih *file ciphertext* yang telah ada atau dengan cara memasukkan teks dalam bentuk binernya ke dalam *textbox.* Setelah kedua *textbox ciphertext* telah diisi kemudian dilanjutkan dengan menekan tombol Hitung untuk melakukan proses penghitungan nilai *avalanche effect.* Setelah proses penghitungan, nilai *avalanche effect* akan ditampilkan dalam bentuk persen dan jumlah bit yang berbeda.

Dalam antarmuka *form* Keygen (Gambar 4.18) terdapat 3 tipe inputan yaitu inputan dalam tipe text, hexadesimal dan biner. Dengan menekan tombol Generate perangkat lunak akan melakukan perubahan pada bit-bit kunci eksternal ini sebesar n > 1 melalui proses *keygen* pada perangkat lunak, dimana n adalah bilangan genap, agar diperoleh kunci eksternal yang lain.



Gambar 4.18 Antarmuka *form* Keygen

4.4 Analisa Hasil

Dari uji coba terhadap perangkat lunak dengan menggunakan kedua jenis algoritma yakni algoritma kriptografi DES dan AES sebagaimana yang telah dijelaskan pada sub bab 3.1. Hasil uji untuk tipe uji coba pertama dengan menggunakan DES dapat dilihat pada Tabel 4.4.

Tabel 4.4 Hasil uji untuk tipe uji coba pertama menggunakan DES

Pasangan *ciphertext*	**∑ bit *ciphertext* berbeda**	**∑ bit *ciphertextA1 atau ciphertext1P***	***Avalanche effect* (%)**
ciphertext 1 & 1	0	448	0
ciphertext 1 & 2	0	448	0
ciphertext 1 & 3	0	448	0
ciphertext 1 & 4	110	448	24
ciphertext 1 & 5	114	448	25
ciphertext 1 & 6	114	448	25
ciphertext 1 & 7	114	448	25

ciphertext 1 & 8	113	448	25
ciphertext 1 & 9	110	448	24
ciphertext 1 & 10	110	448	24
ciphertext 1 & 11	110	448	24
ciphertext 1 & 12	126	448	28
ciphertext 1 & 13	116	448	25
ciphertext 1 & 14	116	448	24
ciphertext 1 & 15	116	448	24
ciphertext 1 & 16	106	448	23
ciphertext 1 & 17	106	448	23
ciphertext 1 & 18	106	448	23
ciphertext 1 & 19	106	448	23
ciphertext 1 & 20	112	448	25

Dari hasil uji untuk tipe uji coba pertama menggunakan DES (Tabel 4.4). Uji pertama membandingkan *ciphertext* 1 hasil enkripsi dari *External Key* pertama (Tabel 4.1) dan *Plaintext* (Subbab 4.2.2) dengan *ciphertext* 1 didapatkan hasil 0 jumlah bit yang berbeda, 448 bit untuk panjang *ciphertext*, 0% untuk nilai *Avalanche Effect*. Hal yang sama didapatkan dari hasil uji coba perbandingan antara *ciphertext* 1 dengan *ciphertext* 2 dan *ciphertext* 1 dengan *ciphertext* 3. Pada uji coba keempat yaitu membandingkan antara *ciphertext* 1 dengan *ciphertext* 4 hasil enkripsi dari *External Key* keempat (Tabel 4.1) dengan *Plaintext* (Subbab 4.2.2) didapatkan hasil 110 jumlah bit yang berbeda, 448 bit untuk panjang *ciphertext*, 24% untuk nilai *Avalanche Effect*. Uji coba dilanjutkan dengan proses yang sama sampai pada perbandingan antara *ciphertext* 1 dengan *ciphertext* 20.

Dapat dilihat bahwa nilai *avalanche effect* terbesar dengan 126 jumlah bit *ciphertext* yang berbeda dengan panjang *ciphertext* 448 bit adalah sebesar 28% (Tabel 4.4). Sedangkan hasil uji untuk tipe uji coba pertama dengan menggunakan AES dapat dilihat pada Tabel 4.5.

Dari hasil uji untuk tipe uji coba pertama menggunakan AES (Tabel 4.5). Uji coba pertama membandingkan *ciphertext* 1 hasil enkripsi dari *External Key* pertama (Tabel 4.2) dan *Plaintext* (Subbab 4.2.2) dengan *ciphertext* 1 didapatkan hasil 0 jumlah bit yang berbeda, 512 untuk panjang bit *ciphertext*, 0% untuk nilai *Avalanche Effect*. Pada uji coba kedua yaitu membandingkan antara *ciphertext* 1 dengan *ciphertext* hasil enkripsi dari *External Key* kedua (Tabel 4.2)

dengan *Plaintext* (Subbab 4.2.2) didapatkan hasil 257 jumlah bit yang berbeda, 512 untuk panjang bit *ciphertext*, 50% untuk nilai *Avalanche Effect*. Uji coba dilanjutkan dengan proses yang sama sampai pada perbandingan antara *ciphertext* 1 dengan *ciphertext* 20.

Dapat dilihat bahwa nilai *avalanche effect* terbesar dengan 273 jumlah bit *ciphertext* yang berbeda dengan panjang *ciphertext* 512 bit adalah sebesar 53% (Tabel 4.5).

Perbandingan nilai *avalanche effect* pada DES dan AES untuk tipe uji coba pertama dapat dilihat pada Tabel 4.6.

Tabel 4.5 Hasil uji untuk tipe uji coba pertama menggunakan AES

Pasangan *ciphertext*	**∑ bit *ciphertext* berbeda**	**∑ bit *ciphertextA1 atau ciphertext1P***	***Avalanche effect* (%)**
ciphertext 1 & 1	0	512	0
ciphertext 1 & 2	257	512	50
ciphertext 1 & 3	257	512	50
ciphertext 1 & 4	266	512	51
ciphertext 1 & 5	253	512	49
ciphertext 1 & 6	267	512	52
ciphertext 1 & 7	250	512	48
ciphertext 1 & 8	273	512	53
ciphertext 1 & 9	255	512	49
ciphertext 1 & 10	259	512	50
ciphertext 1 & 11	261	512	50
ciphertext 1 & 12	251	512	49
ciphertext 1 & 13	251	512	49
ciphertext 1 & 14	244	512	47
ciphertext 1 & 15	273	512	53
ciphertext 1 & 16	255	512	49
ciphertext 1 & 17	258	512	50
ciphertext 1 & 18	261	512	50
ciphertext 1 & 19	253	512	49
ciphertext 1 & 20	259	512	50

Tabel 4.6 Tabel evaluasi tipe uji coba pertama

Jumlah n External key DES	***Avalanche effect DES***	**Jumlah n External key AES**	***Avalanche effect AES***
2	0%	2	50%
4	0%	4	50%
6	24%	6	51%
8	25%	8	49%
10	25%	10	52%
12	25%	12	48%
14	25%	14	53%
16	24%	16	49%
18	24%	18	50%
20	24%	20	50%
22	28%	22	49%
24	25%	24	49%
26	24%	26	47%
28	24%	28	53%
30	23%	30	49%
32	23%	32	50%
34	23%	34	50%
36	23%	36	49%
38	25%	38	50%
Rata-rata	**22%**	**Rata-rata**	**50%**

Dari hasil uji coba pertama terhadap algoritma DES dan AES selanjutnya dilakukan evaluasi terhadap hasil uji coba (Tabel 4.6). Pada algoritma DES *Avalanche Effect* mulai terjadi pada saat terdapat perbedaan sebanyak 6 bit pada *External Key* (Tabel 4.1), sedangkan pada algoritma AES *Avalanche Effect* mulai terjadi pada saat terdapat perbedaan sebanyak 2 bit pada *External Key* (Tabel 4.2).

Dapat diambil kesimpulan bahwa untuk tipe uji coba pertama, rata-rata nilai *avalanche effect* terbesar dipegang oleh algoritma AES yaitu sebesar 50%.

Setelah dilakukan tipe uji coba pertama kemudian dilanjutkan dengan tipe uji coba kedua. Hasil uji untuk tipe uji coba kedua dengan menggunakan algoritma DES dapat dilihat pada Tabel 4.7.

Tabel 4.7 Hasil uji untuk tipe uji coba kedua menggunakan DES

Pasangan *ciphertext*	**∑ bit *ciphertext* berbeda**	**∑ bit *ciphertextA1 atau ciphertext1P***	***Avalanche effect* (%)**
ciphertext 1 & 1	0	448	0
ciphertext 1 & 2	2	448	0
ciphertext 1 & 3	4	448	0
ciphertext 1 & 4	16	448	3
ciphertext 1 & 5	23	448	5
ciphertext 1 & 6	25	448	5
ciphertext 1 & 7	27	448	6
ciphertext 1 & 8	30	448	6
ciphertext 1 & 9	29	448	6
ciphertext 1 & 10	31	448	6
ciphertext 1 & 11	48	448	10
ciphertext 1 & 12	47	448	10
ciphertext 1 & 13	52	448	11
ciphertext 1 & 14	52	448	11
ciphertext 1 & 15	50	448	11
ciphertext 1 & 16	53	448	11
ciphertext 1 & 17	50	448	11
ciphertext 1 & 18	50	448	11
ciphertext 1 & 19	52	448	11
ciphertext 1 & 20	52	448	11

Uji coba kedua menggunakan DES (Tabel 4.7). Uji pertama membandingkan *ciphertext* 1 hasil enkripsi dari *External Key* DES (Subbab 4.2.1) dan *Plaintext* pertama (Tabel 4.3) dengan *ciphertext* 1 didapatkan hasil 0 jumlah bit yang berbeda, 448 bit untuk panjang *ciphertext*, 0% untuk nilai *Avalanche Effect.* Pada uji coba kedua yaitu membandingkan *ciphertext* 1 dengan *ciphertext* hasil enkripsi dari *External Key* DES (Subbab 4.2.1) dan *Plaintext* kedua (Tabel

4.3) didapatkan hasil 2 bit *ciphertext* yang berbeda, 448 bit untuk panjang *ciphertext* dan 0% untuk nilai *Avalanche Effect*. Pada uji coba ketiga yaitu membandingkan *ciphertext* 1 dengan *ciphertext* 3 hasil enkripsi dari *External Key* DES (Subbab 4.2.1) dan *Plaintext* ketiga (Tabel 4.3) didapatkan hasil 4 bit *ciphertext* yang berbeda, 448 bit untuk panjang *ciphertext* dan 0% untuk nilai *Avalanche Effect*. Pada uji coba keempat yaitu membandingkan *ciphertext* 1 dengan *ciphertext* 4 hasil enkripsi dari *External Key* DES (Subbab 4.2.1) dan *Plaintext* keempat (Tabel 4.3) didapatkan hasil 16 bit *ciphertext* yang berbeda, 448 bit untuk panjang *ciphertext* dan 3% untuk nilai *Avalanche Effect*. Uji coba dilanjutkan dengan proses yang sama sampai pada perbandingan antara *ciphertext* 1 dengan *ciphertext* 20.

Dari hasil uji untuk tipe uji coba kedua menggunakan DES (Tabel 4.7) dapat dilihat bahwa nilai *avalanche effect* terbesar dengan 53 jumlah bit *ciphertext* yang berbeda, 448 bit untuk panjang *ciphertext* adalah sebesar 11%. Sedangkan hasil uji untuk tipe uji coba kedua menggunakan AES dapat dilihat pada Tabel 4.8.

Tabel 4.8 Hasil uji untuk tipe uji coba kedua menggunakan AES

Pasangan *ciphertext*	**∑ bit *ciphertext* berbeda**	**∑ bit *ciphertextA1 atau ciphertext1P***	***Avalanche effect* (%)**
ciphertext 1 & 1	0	512	0
ciphertext 1 & 2	67	512	13
ciphertext 1 & 3	64	512	12
ciphertext 1 & 4	64	512	12
ciphertext 1 & 5	74	512	14
ciphertext 1 & 6	61	512	11
ciphertext 1 & 7	65	512	12
ciphertext 1 & 8	64	512	12
ciphertext 1 & 9	70	512	13
ciphertext 1 & 10	135	512	26
ciphertext 1 & 11	137	512	26
ciphertext 1 & 12	123	512	24
ciphertext 1 & 13	133	512	25
ciphertext 1 & 14	128	512	25

ciphertext 1 & 15	123	512	24
ciphertext 1 & 16	135	512	26
ciphertext 1 & 17	147	512	28
ciphertext 1 & 18	134	512	26
ciphertext 1 & 19	119	512	23
ciphertext 1 & 20	139	512	27

Uji coba kedua menggunakan AES (Tabel 4.8). Uji pertama membandingkan *ciphertext* 1 hasil enkripsi dari *External Key* AES (Subbab 4.2.1) dan *Plaintext* pertama (Tabel 4.3) dengan *ciphertext* 1 didapatkan hasil 0 jumlah bit yang berbeda, 512 bit untuk panjang *ciphertext*, 0% untuk nilai *Avalanche Effect.* Pada uji coba kedua yaitu membandingkan *ciphertext* 1 dengan *ciphertext* 2 hasil enkripsi dari *External Key* AES(Subbab 4.2.1) dan *Plaintext* kedua (Tabel 4.3) didapatkan hasil 67 bit *ciphertext* yang berbeda, 512 bit untuk panjang *ciphertext* dan 13% untuk nilai *Avalanche Effect.* Uji coba dilanjutkan dengan proses yang sama sampai pada perbandingan antara *ciphertext* 1 dengan *ciphertext* 20.

Dapat dilihat bahwa nilai *avalanche effect* terbesar dengan 147 jumlah bit yang berbeda pada *ciphertext*, 512 bit untuk panjang *chipertext* adalah sebesar 28%.

Perbandingan nilai *avalanche effect* pada DES dan AES untuk tipe uji coba kedua dapat dilihat pada Tabel 4.9.

Tabel 4.9 Tabel evaluasi tipe uji coba kedua

Jumlah n Plaintext DES	***Avalanche effect DES***	**Jumlah n Plaintext AES**	***Avalanche effect AES***
2	0%	2	13%
4	0%	4	12%
6	3%	6	12%
8	5%	8	14%
10	5%	10	11%
12	6%	12	12%
14	6%	14	12%
16	6%	16	13%
18	6%	18	26%

20	10%	20	26%
22	10%	22	24%
24	11%	24	25%
26	11%	26	25%
28	11%	28	24%
30	11%	30	26%
32	11%	32	28%
34	11%	34	26%
36	11%	36	23%
38	11%	38	27%
Rata-rata	**8%**	**Rata-rata**	**20%**

Dari hasil uji coba kedua terhadap algoritma DES dan AES selanjutnya dilakukan evaluasi terhadap hasil uji coba (Tabel 4.9). Pada algoritma DES *Avalanche Effect* mulai terjadi pada saat terdapat perbedaan sebanyak 6 bit pada *Plaintext* (Tabel 4.3), sedangkan pada algoritma AES *Avalanche Effect* mulai terjadi pada saat terdapat perbedaan sebanyak 2 bit pada *Plaintext* (Tabel 4.3).

Dari Tabel 4.9 dapat diambil kesimpulan bahwa untuk tipe uji coba kedua, nilai rata-rata *avalanche effect* terbesar dipegang oleh algoritma AES dengan nilai *avalanche effect* sebesar 12%.

Dari kedua tipe percobaan yang telah dilakukan, didapatkan kesimpulan bahwa algoritma enkripsi AES menghasilkan nilai *avalanche effect* lebih besar dibandingkan dengan algoritma enkripsi DES. Dengan nilai *avalanche effect* yang besar pada algoritma enkripsi AES, perubahan kecil pada *plaintext* atau *external key* akan memberikan pengaruh yang besar pada *ciphertext*. Selain menghasilkan nilai *avalanche effect* yang lebih besar, algoritma enkripsi AES juga mempunyai bit *ciphertext* yang lebih panjang dibandingkan dengan algoritma enkripsi DES. Sehingga dapat disimpulkan bahwa algoritma enkripsi AES terbukti lebih baik dari algoritma enkripsi DES jika ditinjau dari nilai *avalanche effect* dan panjang bit *ciphertext*.

BAB V
PENUTUP

5.1 Kesimpulan

Kesimpulan yang didapat dari skripsi ini adalah :

1. Pada uji coba 1 yaitu uji coba dengan 1 plaintext dan 20 kunci eksternal yang berbeda didapat hasil algoritma enkripsi AES mampu menghasilkan nilai rata-rata *avalanche effect* sebesar 50% sedangkan algoritma enkripsi DES hanya mampu menghasilkan nilai rata-rata *avalanche effect* sebesar 22%.
2. Pada uji coba 2 yaitu uji coba dengan 1 kunci eksternal dan 20 plaintext yang berbeda didapat hasil algoritma enkripsi AES mampu menghasilkan nilai rata-rata *avalanche effect* sebesar 20% sedangkan algoritma enkripsi DES hanya mampu menghasilkan nilai rata-rata *avalanche effect* sebesar 8%.
3. Perubahan yang terjadi pada bit *External Key* lebih besar pengaruhnya terhadap *Chipertext* yang dihasilkan daripada perubahan yang terjadi pada bit *Plaintext*.
4. Hasil uji coba 1 dan 2 menunjukkan bahwa algoritma enkripsi AES terbukti mempunyai nilai *avalanche effect* yang lebih baik dibandingkan dengan algoritma enkripsi DES.

5.2 Saran

Untuk pengujian lebih lanjut disarankan untuk menguji file teks dengan ukuran file yang lebih bervariasi. Untuk pengujian enkripsi *file text* yang berukuran besar, disarankan penambahan parameter uji waktu proses eksekusi, karena untuk melakukan enkripsi file text yang berukuran besar dibutuhkan waktu yang lama dalam proses eksekusinya.

DAFTAR PUSTAKA

Bakhtiari, S. 1994. *Linear Cryptanalysis of DES Cipher*. Wolongong University. Australia.

Budiyono, A. 2004. *Enkripsi Data Kunci Simetris Dengan Algoritma Kriptografi LOKI97*. ITB. Bandung.

Federal Information Processing Standards Publication. 1999. *Data Encryption Standard (DES)*. U.S. Department Of Commerce/National Institute of Standards and Technology.

Federal Information Processing Standards Publication 197. 2001. *Advanced Encryption Standard (AES)*. U.S. Department Of Commerce/National Institute of Standards and Technology.

Gladman, B Dr. *A Specification for Rijndael, the AES Algorithm v3.1*.

Grabbe, O., J. *The DES Algorithm Illustrated*. http://www.aci.net/Kalliste/des.htm. Diakses tanggal 13 Februari 2008.

Kurniawan, Y., C. 2007. *Penerapan Algoritma Kriptografi DES, AES dan RSA serta Algoritma Kompresi LZW untuk Pengamanan Berkas Digital*. Universitas Kristen Petra. Surabaya.

Munir, Rinaldi. 2006. *Kriptografi*. Informatika Bandung. Bandung.

Raharjo, B. 1998. *Keamanan Sistem Informasi Berbasis Internet.* http://budi.insan.co.id/courses/security/docs/handbook.pdf. Diakses pada tanggal 9 Februari 2007.

Schneier, B. 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* 2nd ed. John Wiley and Sons. New Jersey.

Stallings, W. 2005. *Cryptography and Network Security Principles and Practices, Fourth Edition*. Prentice Hall.

http://www.codeproject.com/kb/reciper/simple_cryptographer.
Diakses tanggal 16 Oktober 2008.