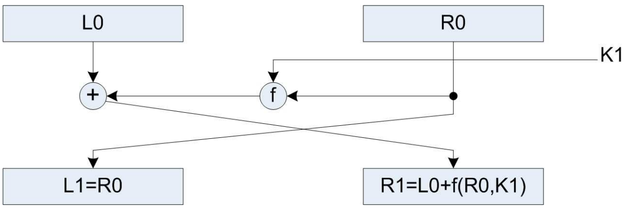
Bab 8

Analisa Block Cipher

Enkripsi *block cipher* menggunakan fungsi *cipher* yang agak lemah berulang kali untuk mendapatkan fungsi *cipher* yang kuat. Setiap putaran menambahkan efek *confusion* dan *diffusion* terhadap proses enkripsi. Semakin banyak putaran yang digunakan, semakin besar efek *confusion* dan *diffusion* dalam proses enkripsi, selama efek *confusion* dan *diffusion* belum "jenuh." Jadi, agar optimal, jumlah putaran adalah jumlah terkecil yang mengakibatkan efek *confusion* dan *diffusion* menjadi "jenuh."



Gambar 8.1: 1 Putaran DES

Mari kita ambil contoh enkripsi DES. Gambar 8.1 menunjukkan 1 putaran dalam DES. Kita coba dapatkan kunci putaran K_1 jika L_0, R_0 dan L_1, R_1 diketahui. Berdasarkan rumus

$$R_1 = L_0 \oplus f(R_0, K_1)$$

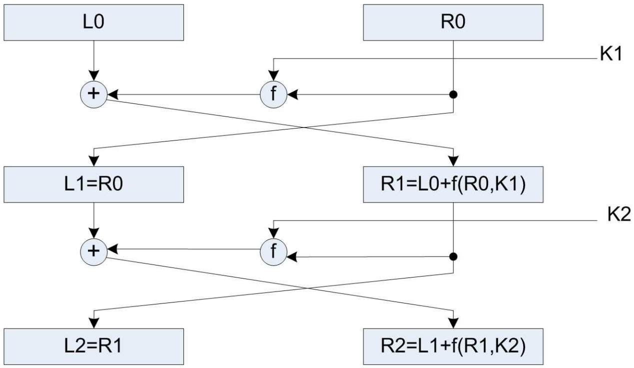
kita dapat mengkalkulasi output fungsi *cipher* f:

$$f(R_0, K_1) = R_1 \oplus L_0.$$

Gambar 7.3 menunjukkan fungsi *cipher* f untuk DES. Jika output fungsi *cipher* f diketahui, maka output dari S-boxes S1 sampai dengan S8 dapat dikalkulasi menggunakan *inverse* permutasi P. Untuk setiap S-box, jika output diketahui, maka ada 4 kandidat untuk input S-box. Jadi untuk 8 S-boxes, ada 4^8 atau 2^{16} kandidat input S-boxes yang dapat menghasilkan output yang diketahui. Menggunakan rumus

$$K_1 = E(R_0) \oplus S_I$$

dimana S_I adalah kandidat input S-boxes, terdapat 2^{16} kandidat kunci putaran K_1.



Gambar 8.2: 2 Putaran DES

Dengan 2 putaran DES, jika L_0, R_0 dan L_2, R_2 diketahui, maka kita dapat gunakan rumus

$$\begin{aligned} f(R_0, K_1) &= L_0 \oplus R_1 \\ &= L_0 \oplus L_2 \end{aligned}$$

untuk mencari kandidat K_1 dan rumus

$$\begin{aligned} f(L_2, K_2) &= f(R_1, K_2) \\ &= L_1 \oplus R_2 \\ &= R_0 \oplus R_2 \end{aligned}$$

untuk mencari kandidat K_2. Seperti dengan 1 putaran, terdapat 2^{16} kandidat kunci putaran K_1. Untuk kunci putaran K_2 juga terdapat 2^{16} kandidat. Jika dependensi K_2 terhadap K_1 tidak digunakan maka kita harus mencoba 2^{32} kombinasi K_1 dan K_2, yang masih jauh lebih kecil dari jumlah kemungkinan kunci DES yaitu 2^{56}.

Sampai dengan 2 putaran, efek *confusion* dan *diffusion* belum terlalu besar sehingga masih dapat dianalisa secara naif. Akan tetapi untuk lebih dari

2 putaran, efek *confusion* dan *diffusion* semakin besar dan analisa menjadi semakin rumit. Nilai L dan R untuk putaran bukan pertama dan bukan terahir tidak dapat ditentukan dengan pasti dari nilai L dan R untuk putaran pertama dan terahir. Oleh karena itu diperlukan analisa yang lebih canggih seperti *differential cryptanalysis* atau *linear cryptanalysis*.

Cryptanalysis kerap digunakan untuk mencari kelemahan algoritma enkripsi. Tetapi, meskipun suatu algoritma dianggap dapat "dipecahkan" dengan *cryptanalysis*, ini belum tentu berarti bahwa pemecahan dapat dipraktekkan untuk mencari kunci enkripsi. Ini hanya berarti bahwa algoritma mempunyai "kelemahan." Sebagai contoh, dengan *linear cryptanalysis*, DES memerlukan 2^{47} pencarian menggunakan *known plaintext attack* dimana kita harus mempunyai naskah acak untuk 2^{47} naskah yang diketahui (secara *brute force* DES memerlukan 2^{55} pencarian). Akan tetapi dalam prakteknya sangat mustahil kita bisa mendapatkan sedemikian banyak naskah acak yang dienkripsi "musuh" untuk naskah yang kita ketahui.

8.1 Differential Cryptanalysis

Secara garis besar, *differential cryptanalysis* adalah teknik untuk mencari kunci enkripsi *block cipher* dari analisa efek perbedaan naskah asli terhadap perbedaan naskah acak. *Differential cryptanalysis* ditemukan oleh Eli Biham dan Adi Shamir, dan dalam bentuk dasarnya merupakan apa yang disebut *chosen-plaintext attack*. Namun jika cukup banyak naskah asli (*plaintext*) yang diketahui, maka teknik ini dapat digunakan untuk *known-plaintext attack* (lihat bagian 2.3.1).

Teknik *differential cryptanalysis* pada awalnya digunakan untuk menganalisa DES, dan DES dijadikan contoh untuk menjelaskan *differential cryptanalysis*. Pembaca dapat meninjau kembali bagian 7.1 mengenai DES. Disini kami hanya akan menjelaskan konsep-konsep dasar dari *differential cryptanalysis*. Bagi pembaca yang ingin memperdalam pengetahuan mengenai *differential cryptanalysis*, dianjurkan untuk membaca [bih91].

Konsep perbedaan dalam *differential cryptanalysis* dirumuskan dengan operasi *exclusive or*. Jadi perbedaan antara dua naskah asli P_1 dan P_2 adalah

$$P_1 \oplus P_2$$

dimana operasi *exclusive or* dilakukan secara *bitwise*. Jika C_1 dan C_2 adalah naskah acak untuk P_1 dan P_2, maka efek $P_1 \oplus P_2$ terhadap $C_1 \oplus C_2$ dapat memberikan informasi mengenai kunci enkripsi. Analisa mencoba mengeksploitasi kecenderungan fungsi *cipher* dan didasarkan pada sifat aljabar operasi *exclusive or*.

Efek dari permutasi seperti *initial permutation* (IP) adalah linear dengan

$$IP(P_1) \oplus IP(P_2) = IP(P_1 \oplus P_2),$$

jadi efek permutasi terhadap perbedaan tidak terlalu rumit. Permutasi yang dilakukan diluar putaran seperti IP dan IP^{-1} sama sekali tidak mempersulit analisa.

Mari kita lihat efek dari fungsi *cipher* f yang beroperasi terhadap setengah dari naskah sebesar 32 bit. Efek dari ekspansi E juga linear dengan

$$E(P_1) \oplus E(P_2) = E(P_1 \oplus P_2),$$

jadi ekspansi juga tidak membuat rumit perbedaan, jadi tidak mempengaruhi analisa satu putaran. Akan tetapi, ekspansi, yang selain mengekspansi juga melakukan permutasi, mempengaruhi tingkat kesulitan analisa lebih dari dua putaran karena efek *avalanche* yang ditimbulkannya. Efek *avalanche* terjadi karena perbedaan 1 bit dalam input setelah melewati S-box menjadi perbedaan sedikitnya 2 bit. Karena efek ekspansi, perbedaan 2 bit akan menjadi input 3 S-boxes dua putaran kemudian yang oleh 3 S-boxes dijadikan perbedaan 6 bit, dan seterusnya. Jadi dengan setiap putaran, efek perbedaan semakin besar bagaikan *avalanche*.

Efek dari operasi *exclusive or* dengan kunci putaran adalah

$$(P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

yang berarti tidak ada efek terhadap perbedaan. Efek dari permutasi P juga linear, jadi yang sangat menentukan dalam *differential cryptanalysis* adalah efek dari substitusi S-box yang diketahui sebagai tidak linear.

Penjelasan konsep-konsep dasar *differential cryptanalysis* kami bagi menjadi 3 bagian:

1. Analisa 1 putaran.
2. Mekanisme n-round characteristic.
3. Penggunaan n-round characteristic.

8.1.1 Analisa 1 Putaran

Bagaimana kita mencari bits kunci putaran dari XOR pasangan input dan XOR pasangan output suatu S-box? Sebagai contoh kita umpamakan bahwa XOR pasangan input adalah $0x34$ (hexadecimal 34) dan XOR pasangan output adalah $0xd$ (hexadecimal d) dan S-box adalah S1. (Kita gunakan notasi $0x34 \rightarrow 0xd$ untuk menandakan bahwa XOR input $0x34$ dapat menghasilkan XOR output $0xD$.) Kita umpamakan juga bahwa bits pasangan hasil ekspansi E adalah $0x35$ dan $0x01$. Bits input untuk S1 didapat dari XOR bits hasil ekspansi E dengan bits kunci k_1. Jadi pasangan input S1, sebut saja x dan y mempunyai rumus:

$$\begin{aligned} x &= 0x35 \oplus k_1, \\ y &= 0x01 \oplus k_1, \end{aligned}$$

yang berarti

$$k_1 = 0x35 \oplus x = 0x01 \oplus y.$$

Jadi bits kunci putaran didapat dari XOR pasangan hasil ekspansi dengan pasangan input S1. Namun tidak semua pasangan input S1 dapat menghasilkan $0xd$ sebagai XOR output S1. Hanya ada 8 pasangan input x dan y dengan XOR $0x34$ yang menghasilkan XOR output $0xd$, oleh karena itu hanya ada 8 kandidat nilai bits kunci yang dimungkinkan seperti terlihat dalam tabel 8.1.

Pasangan input S1	Bits kunci putaran
$0x06$, $0x32$	$0x33$
$0x32$, $0x06$	$0x07$
$0x10$, $0x24$	$0x25$
$0x24$, $0x10$	$0x11$
$0x16$, $0x22$	$0x23$
$0x22$, $0x16$	$0x17$
$0x1c$, $0x28$	$0x29$
$0x28$, $0x1c$	$0x1d$

Tabel 8.1: Bits kunci untuk $0x34 \rightarrow 0xd$ dan ekspansi ($0x35$, $0x01$).

Jadi dengan menganalisa hasil transformasi S1 terhadap pasangan ekspansi $0x35$ dan $0x01$, ruang pencarian bits kunci putaran diperkecil dari 64 kandidat menjadi 8 kandidat.

Jika kita mempunyai pasangan ekspansi lain (mungkin dengan hasil XOR yang berbeda) yang menghasilkan tabel lain, kita dapat memperoleh informasi tambahan mengenai bits kunci putaran. Bits kunci putaran harus berada dalam semua tabel yang dihasilkan, jadi setelah mendapatkan tabel 8.2, kandidat untuk bits kunci putaran tinggal dua yaitu

$$0x23 \text{ dan } 0x17.$$

Analisa dapat dilanjutkan menggunakan pasangan ekspansi lainnya sampai kandidat bits kunci putaran tinggal satu sehingga bits kunci putaran dapat ditentukan.

Jika proses pencarian bits kunci putaran menggunakan analisa efek S-box tidak selesai, hasil analisa dapat digunakan untuk menentukan probabilitas berbagai kandidat bits kunci putaran. Pendekatan probabilistik inilah sebenarnya yang digunakan dalam *differential cryptanalysis*.

Secara garis besar, metode yang digunakan *differential cryptanalysis* untuk mencari kunci putaran adalah sebagai berikut:

1. Kita pilih XOR untuk naskah asli.

Pasangan input S1	Bits kunci putaran
$0x01$, $0x35$	$0x20$
$0x35$, $0x01$	$0x14$
$0x02$, $0x36$	$0x23$
$0x36$, $0x02$	$0x17$
$0x15$, $0x21$	$0x34$
$0x21$, $0x15$	$0x00$

Tabel 8.2: Bits kunci untuk $0x34 \rightarrow 0x3$ dan ekspansi ($0x21$, $0x15$).

2. Kita buat beberapa pasangan naskah asli dengan XOR yang dipilih, kita lakukan enkripsi terhadap pasangan, dan simpan pasangan terenkripsi.
3. Untuk setiap pasangan, cari XOR output yang diharapkan untuk sebanyak mungkin S-boxes untuk putaran terahir dari XOR naskah asli dan pasangan terenkripsi (XOR input fungsi *cipher* f untuk putaran terahir diketahui karena merupakan XOR bagian dari pasangan terenkripsi).
4. Untuk setiap kandidat kunci putaran, hitung pasangan yang menghasilkan XOR yang diharapkan jika menggunakan kandidat kunci putaran.
5. Kunci putaran yang terpilih adalah kandidat kunci putaran yang mempunyai hitungan terbesar.

8.1.2 Mekanisme n-round Characteristic

Sebelum melanjutkan penjelasan mekanisme yang digunakan, kita bahas dahulu model probabilistik yang digunakan dalam *differential cryptanalysis*. Untuk input, semua input yang dimungkinkan mempunyai probabilitas yang sama. Demikian juga dengan kunci putaran, semua kunci putaran mempunyai probabilitas a priori yang sama dan setiap kunci putaran adalah independen dari semua kunci putaran sebelumnya (jadi algoritma untuk *key scheduling* tidak berperan dalam model probabilistik yang digunakan). Meskipun kunci putaran DES sebenarnya tidak independen (algoritma *key scheduling* membuat relasi antar kunci putaran deterministik), asumsi ini menyederhanakan model probabilistik yang digunakan, tetapi ruang pencarian menjadi lebih besar. Dengan asumsi kunci putaran independen terdapat 2^{768} kombinasi kunci putaran untuk 16 putaran DES, sedangkan DES sebenarnya hanya mempunyai 2^{56} kombinasi kunci putaran. Tentunya dalam praktek *differential cryptanalysis*, pengetahuan mengenai algoritma *key scheduling* dapat digunakan untuk mempermudah analisa secara keseluruhan.

Kita mulai penjelasan mekanisme pada tingkat S-box. Setiap S-box mempunyai input 6 bit dan output 4 bit, jadi ada 64 nilai XOR input dan 16 nilai

XOR output. Setiap nilai 6 bit XOR input dapat berasal dari 64 pasangan yang berbeda, sebagai contoh

$$\begin{array}{ccc} 000010 & \oplus & 000001, \\ 000001 & \oplus & 000010, \\ 000110 & \oplus & 000101, \\ 000101 & \oplus & 000110 \end{array}$$

dan ada 60 pasangan lainnya semua menghasilkan 000011 sebagai XOR input. Kecuali jika nilai XOR input adalah 000000, S-box dapat menghasilkan nilai XOR output yang berbeda untuk nilai XOR input yang sama tetapi dari pasangan yang berbeda. Sebagai contoh, dengan nilai $0x03$ (000011) sebagai XOR input, S1 akan menghasilkan XOR output $0x4$ (0100) untuk pasangan input $0x01$ dan $0x02$ sedangkan untuk pasangan input $0x21$ dan $0x22$ S1 akan menghasilkan XOR output $0xe$ (1110). Tabel 8.3 menunjukkan distribusi XOR output untuk S1 untuk nilai XOR input $0x00$, $0x01$, $0x02$ dan $0x03$ ($0x03$ berarti 03 dalam notasi hexadecimal, yang dalam notasi biner menjadi 000011). Semua pasangan input dengan $0x00$ sebagai nilai XOR input (ada 64 pasangan)

XOR	XOR output															
input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
02	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
03	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
...	...															

Tabel 8.3: Tabel parsial distribusi XOR output untuk S1

menghasilkan $0x0$ sebagai XOR output. Untuk nilai $0x01$ sebagai XOR input, 6 pasangan input menghasilkan $0x3$ sebagai XOR output, 2 pasangan input menghasilkan $0x5$, 4 pasangan input menghasilkan $0x6$, dan tidak ada pasangan input yang dapat menghasilkan $0x0$, $0x1$, $0x2$, $0x4$ dan $0x8$. Secara rerata, hanya sekitar 80 persen dari nilai XOR output dimungkinkan oleh S1 dan S-box lainnya untuk setiap nilai XOR input. Tabel lengkap untuk distribusi XOR untuk semua S-box DES terdapat dalam [bih91], dan tabel juga dapat dikalkulasi berdasarkan spesifikasi S-box DES.

Jika suatu nilai XOR input, sebut saja X, dapat menghasilkan suatu nilai XOR output, sebut saja Y, untuk suatu S-box, maka kita katakan bahwa X dapat menyebabkan Y ($X \rightarrow Y$) dengan probabilitas $\frac{n_i}{64}$ dimana n_i adalah jumlah pasangan dengan XOR input X yang menghasilkan XOR output Y (n_i dapat diambil dari tabel distribusi XOR).

Konsep “menyebabkan” ($\rightarrow$) juga berlaku untuk transformasi fungsi *cipher* f dimana besar input dan output adalah 32 bit. Dengan X dan Y masing-masing sebesar 32 bit, kita katakan bahwa $X \rightarrow Y$ menggunakan fungsi *cipher* f dengan probabilitas

$$p = \frac{n_Y}{n}$$

dimana

- n_Y adalah jumlah semua kombinasi pasangan input dan kunci putaran dengan XOR input X dan menghasilkan XOR output Y, dan
- n adalah jumlah semua kombinasi pasangan input dan kunci putaran dengan XOR input X.

Ada 2^{32} pasangan input dengan XOR X karena input 32 bit, dan ada 2^{48} kunci putaran karena kunci putaran 48 bit, oleh sebab itu $n = 2^{80}$, jadi

$$p = \frac{n_Y}{2^{80}}.$$

Teorema 28 *Untuk DES, jika $X \rightarrow Y$ menggunakan fungsi cipher f dengan probabilitas p, maka untuk setiap pasangan input dengan XOR X, p juga merupakan probabilitas bahwa pasangan input akan menghasilkan XOR output Y. Jadi p adalah rasio jumlah kunci putaran yang mengakibatkan fungsi cipher f menghasilkan pasangan output dengan XOR Y, dibagi dengan jumlah semua kunci putaran.*

Untuk membuktikan teorema 28, kita mengetahui bahwa kunci putaran tidak mengubah XOR hasil ekspansi, jadi XOR input S-boxes sama dengan XOR hasil ekspansi. Kunci putaran mengubah pasangan tetapi tetap mempertahankan XOR pasangan. Jika i_1, i_2 adalah pasangan input fungsi *cipher* f dengan

$$i_1 \oplus i_2 = X$$

dan

$$\begin{aligned} e_1 &= E(i_1), \\ e_2 &= E(i_2) \end{aligned}$$

dan s_1, s_2 adalah pasangan input S-boxes yang menghasilkan pasangan o_1, o_2 sebagai output fungsi *cipher* f dengan

$$o_1 \oplus o_2 = Y$$

dan

$$s_1 \oplus s_2 = e_1 \oplus e_2$$

maka kunci putaran

$$k = e_1 \oplus s_1$$

dapat digunakan sehingga

$$\begin{aligned} s_1 &= e_1 \oplus k \text{ dan} \\ s_2 &= e_2 \oplus k. \end{aligned}$$

Karena pasangan output o_1, o_2 hanya tergantung pada pasangan s_1, s_2, setiap pasangan output dengan XOR Y mempunyai satu kunci putaran yang menghasilkan pasangan tersebut dari i_1, i_2. Karena setiap pasangan menggunakan kunci yang berbeda, banyaknya kunci putaran yang menghasilkan pasangan output dengan XOR Y dari pasangan i_1, i_2 sama dengan banyaknya pasangan output dengan XOR Y yang dapat dihasilkan dari pasangan i_1, i_2. Jadi

$$p = \frac{n_k}{2^{48}}$$

dimana n_k adalah banyaknya kunci putaran yang menghasilkan pasangan output dengan XOR Y dari pasangan i_1, i_2. Probabilitas ini konstan untuk semua pasangan input dengan XOR X jadi sama dengan probabilitas untuk semua pasangan input dengan XOR X secara rerata.

Kita dapat gabungkan probabilitas kedelapan S-box sebagai berikut. Jika

$$X_i \rightarrow Y_i \text{ dengan probabilitas } p_i$$

untuk $1 \leq i \leq 8$, dimana

$$\begin{aligned} X_1X_2X_3X_4X_5X_6X_7X_8 &= E(X) \text{ dan} \\ P(Y_1Y_2Y_3Y_4Y_5Y_6Y_7Y_8) &= Y \end{aligned}$$

maka

$$X \rightarrow Y$$

dengan probabilitas $p_1p_2p_3p_4p_5p_6p_7p_8$, jadi probabilitas untuk setiap S-box dikalikan untuk mendapatkan probabilitas untuk fungsi *cipher* f.

Mekanisme untuk menggabungkan hasil putaran (butir 3 sampai dengan 5 pencarian kunci putaran) menjadi hasil dari semua putaran dinamakan *n-round characteristic* oleh Biham dan Shamir. Suatu *n-round characteristic* Ω adalah

$$\Omega = (\Omega_P, \Omega_\Lambda, \Omega_T)$$

dimana

- Ω_P adalah XOR dari naskah asli sebesar m bit (64 bit untuk DES),

- $\Omega_\Lambda = (\Lambda_1, \Lambda_2, \ldots \Lambda_n)$,
  $\Lambda_i = (\lambda_I^i, \lambda_O^i)$ untuk $1 \leq i \leq n$,
  λ_I^i adalah XOR input fungsi *cipher* f sebesar $\frac{m}{2}$ bit untuk putaran i,
  λ_O^i adalah XOR output fungsi *cipher* f sebesar $\frac{m}{2}$ bit untuk putaran i, dan

- Ω_T adalah XOR dari naskah acak sebesar m bit.

Untuk setiap *n-round characteristic* Ω,

$$\begin{aligned} \lambda_I^1 &= \text{belahan kanan dari } \Omega_P, \\ \lambda_I^2 &= \text{belahan kiri dari } \Omega_P \oplus \lambda_O^1, \\ \lambda_I^n &= \text{belahan kanan dari } \Omega_T, \\ \lambda_I^{n-1} &= \text{belahan kiri dari } \Omega_T \oplus \lambda_O^n, \text{ dan} \\ \lambda_O^i &= \lambda_I^{i-1} \oplus \lambda_I^{i+1} \text{ untuk } 2 \leq i \leq n-1. \end{aligned}$$

Suatu *n-round characteristic* mempunyai probabilitas bahwa sembarang pasangan naskah asli yang mempunyai XOR sama dengan Ω_P memenuhi *n-round characteristic*, dengan asumsi sembarang kunci putaran yang independen digunakan untuk setiap putaran. Sebagai contoh, *1-round characteristic*

$$((L, 0), ((0, 0)), (L, 0))$$

mempunyai probabilitas 1 (dengan sembarang L), dan merupakan satu-satunya *n-round characteristic* yang mempunyai probabilitas lebih dari $\frac{1}{4}$.

Probabilitas suatu putaran i adalah probabilitas dari Λ_i (dengan notasi $P(\Lambda_i)$), yaitu probabilitas λ_I^i akan menghasilkan λ_O^i. Probabilitas ini telah dibahas dan menjadi subyek dari teorema 28. Probabilitas ini sama dengan probabilitas untuk

$$E(\lambda_I^i) \rightarrow P^{-1}(\lambda_O^i)$$

dimana E adalah ekspansi dalam fungsi *cipher* f dan P^{-1} adalah *inverse* dari permutasi P.

Probabilitas untuk *n-round characteristic* adalah

$$P(\Omega) = P(\Lambda_1)P(\Lambda_2)\ldots P(\Lambda_n).$$

Jadi untuk $\Omega = ((L, 0), ((0, 0)), (L, 0))$

$$P(\Omega) = P(0, 0) = 1$$

karena probabilitas 0 akan menghasilkan 0 adalah 1 (jika tidak ada perbedaan dalam input maka dapat dipastikan tidak akan ada perbedaan dalam output).

Suatu pasangan input disebut *right pair* untuk *n-round characteristic* Ω dan kunci independen K (terdiri dari n kunci putaran yang independen), jika XOR dari pasangan tersebut adalah Ω_P, dan untuk n putaran menggunakan kunci independen K, XOR input putaran i adalah λ_I^i dan XOR output fungsi *cipher* F adalah λ_O^i sesuai dengan Ω.

Teorema 29 *Probabilitas untuk characteristic Ω yang telah didefinisikan merupakan probabilitas aktual bahwa suatu pasangan input dengan XOR Ω_P adalah right pair jika sembarang kunci independen digunakan.*

Untuk membuktikan teorema 29, probabilitas bahwa pasangan input dengan XOR Ω_P adalah *right pair* merupakan probabilitas bahwa untuk setiap putaran i: $\lambda_I^i \to \lambda_O^i$. Probabilitas untuk setiap putaran bersifat independen dari bentuk persisnya pasangan input (asalkan menghasilkan XOR Ω_P, ini dibuktikan oleh teorema 28) dan independen dari apa yang telah terjadi di putaran sebelumnya (karena kunci independen mengacak input ke S-boxes, meskipun XOR input S-boxes tetap sama). Jadi probabilitas bahwa pasangan input merupakan *right pair* merupakan produk dari setiap probabilitas $\lambda_I^i \to \lambda_O^i$, jadi merupakan probabilitas untuk *characteristic* Ω.

8.1.3 Penggunaan n-round Characteristic

Untuk analisa 1 putaran (lihat bagian 8.1.1), kunci putaran dapat dicari dari perpotongan himpunan-himpunan kunci kandidat yang dimungkinkan berbagai pasangan input dan pasangan output. Untuk *n-round characteristic*, ini tidak dapat dilakukan karena perpotongan himpunan-himpunan tersebut biasanya kosong (himpunan yang dihasilkan pasangan input yang bukan *right pair* mungkin tidak memiliki kunci putaran yang sebenarnya sebagai elemen).

Menurut teorema 29, setiap pasangan input yang berupa *right pair*, yang muncul dengan probabilitas *characteristic*, akan menghasilkan kunci putaran yang sebenarnya sebagai kandidat. Kandidat lainnya terdistribusi secara acak, jadi jika semua kandidat dihitung kemunculannya, diharapkan kandidat dengan hitungan terbesar merupakan kunci putaran yang sebenarnya. Kunci putaran yang sebenarnya bagaikan *signal* sedangkan kandidat lainnya adalah *noise*. Semakin tinggi *signal-to-noise ratio*, semakin mudah analisa yang dibutuhkan, dimana *signal-to-noise ratio* S/N didefinisikan sebagai berikut:

$$S/N = \frac{\text{jumlah pasangan yang merupakan } \textit{right pair}}{\text{rerata hitungan per kandidat}}$$

Untuk mendapatkan kunci putaran yang sebenarnya, kita membutuhkan probabilitas *characteristic* yang cukup tinggi, dan pasangan input dan pasangan output yang cukup banyak untuk menjamin cukupnya jumlah pasangan input yang merupakan *right pair*. Dalam prakteknya, *n-round characteristic* dapat digunakan untuk mencari sebagian bits dari kunci putaran, tidak harus sekaligus mencari semua bits kunci putaran. Banyaknya pasangan yang diperlukan tergantung pada jumlah bits kunci putaran yang dicari (k), probabilitas *characteristic* (p), dan jumlah pasangan yang dapat diabaikan karena bukan merupakan *right pairs*. Jika m adalah jumlah pasangan, α adalah rerata hitungan untuk semua pasangan yang dihitung, dan β adalah rasio pasangan yang dihitung dibandingkan dengan total pasangan, karena ada 2^k kandidat yang dihitung, maka rerata hitungan untuk setiap kandidat adalah

$$\frac{m \cdot \alpha \cdot \beta}{2^k}.$$

Rumus untuk S/N menjadi

$$\begin{aligned} S/N &= \frac{m \cdot p}{m \cdot \alpha \cdot \beta / 2^k} \\ &= \frac{2^k \cdot p}{\alpha \cdot \beta}. \end{aligned}$$

Jumlah pasangan *right pair* yang diperlukan untuk mendapatkan kunci putaran yang benar tergantung pada *signal-to-noise ratio*. Hasil empiris menurut Biham dan Shamir menunjukkan untuk rasio S/N sekitar $1-2$, 20 s/d 40 pasangan *right pair* diperlukan. Untuk rasio S/N jauh lebih besar, hanya 3 atau 4 pasangan *right pair* diperlukan. Untuk rasio S/N jauh lebih kecil, pasangan *right pair* yang diperlukan terlalu banyak sehingga tidak praktis.

Dalam penggunaannya, jumlah putaran n dalam *n-round characteristic* tidak harus sama dengan jumlah putaran dalam algoritma enkripsi yang dianalisa. Biasanya, untuk algoritma dengan m putaran,

$$n < m.$$

Sebagai contoh, Biham dan Shamir menggunakan *1-round characteristic* $\Omega = ((0x20000000, 0), ((0, 0)), (0x20000000, 0))$ untuk menganalisa DES yang diperlemah dari 16 putaran menjadi 4 putaran. Pembaca yang ingin melihat berbagai contoh *differential cryptanalysis* dipersilahkan untuk membaca [bih91].

8.1.4 Differential Cryptanalysis DES

Biham dan Shamir menggunakan teknik *differential cryptanalysis* terhadap DES dengan berbagai putaran. Tabel 8.4 memperlihatkan hasil dari analisa mereka. Untuk DES dengan 16 putaran (DES penuh), *differential cryptanalysis* lebih sukar daripada *brute force search*, jadi DES tahan terhadap *differential cryptanalysis*.

Putaran	Kompleksitas
4	2^4
6	2^8
8	2^{16}
9	2^{26}
10	2^{35}
11	2^{36}
12	2^{43}
13	2^{44}
14	2^{51}
15	2^{52}
16	2^{58}

Tabel 8.4: Hasil Differential Cryptanalysis DES

8.2 Linear Cryptanalysis

Linear cryptanalysis adalah teknik memecahkan enkripsi dengan cara membuat perkiraan linear untuk algoritma enkripsi. Kita harus mencari persamaan dalam bentuk

$$P[i_1]\oplus P[i_2]\oplus\ldots\oplus P[i_a]\oplus C[j_1]\oplus C[j_2]\oplus\ldots\oplus C[j_b] = K[k_1]\oplus K[k_2]\oplus\ldots\oplus K[k_c] \quad (8.1)$$

dimana $i_1, i_2 \ldots i_a$, $j_1, j_2 \ldots j_b$, dan $k_1, k_2 \ldots k_c$ adalah posisi bits tertentu, dan persamaan 8.1 mempunyai probabilitas $p \neq 0.5$ untuk sembarang naskah asli P dengan naskah acaknya C. Besar dari $|p - 0.5|$ merepresentasikan efektivitas dari persamaan 8.1. Setelah persamaan 8.1 ditemukan, kita dapat mencari informasi ekuivalen dengan satu bit kunci $K[k_1] \oplus K[k_2] \ldots K[k_c]$ menggunakan algoritma berdasarkan metode kemungkinan maksimal (*maximal likelihood method*) sebagai berikut (Algoritma 1):

1. Kita gunakan T sebagai representasi berapa kali ekspresi sebelah kiri persamaan 8.1 sama dengan 0.

2. Jika $T > N/2$ dimana N adalah jumlah naskah asli yang digunakan, maka kita perkirakan $K[k_1] \oplus K[k_2] \ldots K[k_c] = 0$ (jika $p > 0.5$) atau 1 (jika $p < 0.5$). Jika tidak maka kita perkirakan $K[k_1]\oplus K[k_2] \ldots K[k_c] = 1$ (jika $p > 0.5$) atau 0 (jika $p < 0.5$).

Tingkat kesuksesan metode ini jelas akan meningkat jika N atau $|p - 0.5|$ meningkat. Jadi persamaan yang paling efektif adalah persamaan yang mempunyai nilai maksimal untuk $|p - 0.5|$. Bagaimana kita mencari persamaan yang efektif? Dalam papernya [mat93], Mitsuru Matsui memberi contoh bagaimana menemukan persamaan efektif untuk enkripsi DES. Contoh tersebut akan dicoba untuk dijelaskan disini.

Known-plaintext attack terhadap DES dapat dilakukan dengan persamaan paling efektif untuk $n-1$ putaran, jadi putaran terahir dianggap telah didekripsi menggunakan K_n dan persamaan dengan probabilitas terbaik ($|p-0.5|$ maksimal) dan mengandung F (fungsi *cipher f* DES) adalah sebagai berikut:

$$P[i_1, i_2, \ldots, i_a] \oplus C[j_1, j_2, \ldots, j_b] \oplus F_n(C_L, K_n)[l_1, l_2, \ldots, l_d] = K[k_1, k_2, \ldots, k_c] \tag{8.2}$$

dimana $P[i_1, i_2, \ldots, i_a]$ adalah singkatan untuk $P[i_1] \oplus P[i_2] \oplus \ldots \oplus P[i_a]$.

Efektivitas persamaan 8.2 tergantung pada pilihan untuk K_n (jika K_n yang salah dipilih maka efektivitas persamaan menurun drastis). Jadi metode kemungkinan maksimal dapat digunakan sebagai berikut (Algoritma 2):

1. Untuk setiap kandidat $K_n^{(i)}$ $(i = 1, 2, \ldots)$ dari K_n, kita gunakan T_i untuk merepresentasikan jumlah naskah asli yang mengakibatkan sebelah kiri dari persamaan 8.2 menjadi 0.

2. Jika T_{max} adalah nilai maksimal dari semua T_i dan T_{min} adalah nilai minimal dari semua T_i, maka

   - Jika $|T_{max} - N/2| > |T_{min} - N/2|$ maka kita pilih kandidat kunci yang mengakibatkan T_{max} dan kita pilih $K[k_1, k_2, \ldots k_c] = 0$ (jika $p > 0.5$) atau 1 (jika $p < 0.5$).
   - Jika $|T_{max} - N/2| < |T_{min} - N/2|$ maka kita pilih kandidat kunci yang mengakibatkan T_{min} dan kita pilih $K[k_1, k_2, \ldots k_c] = 1$ (jika $p > 0.5$) atau 0 (jika $p < 0.5$).

8.2.1 Perkiraan Linear untuk S-boxes

Untuk mendapatkan persamaan yang optimal dengan bentuk persamaan 8.2, kita analisa terlebih dahulu perkiraan linear untuk S-boxes. Ini dilakukan dengan mencari kecenderungan pada S-boxes.

Untuk setiap S-box S_a dengan $(a = 1, 2, \ldots, 8), 1 \leq \alpha \leq 63$ dan $1 \leq \beta \leq 15$, kita definisikan $NS_a(\alpha, \beta)$ sebagai berapa kali dari 64 kemungkinan pola input S_a XOR dari bits input yang di*mask* terlebih dahulu menggunakan α, sama dengan XOR dari bits output yang di*mask* terlebih dahulu menggunakan β. Jadi:

$$NS_a(\alpha, \beta) = \sharp\{x | 0 \leq x < 64, (\bigoplus_{s=0}^{5}(x[s] \bullet \alpha[s])) = (\bigoplus_{t=0}^{3}(S_a(x)[t] \bullet \beta[t]))\}$$

dimana $\bullet$ adalah *bitwise and* (yang digunakan untuk proses *masking*) dan $\sharp$ adalah simbol untuk *cardinality* (besar dari himpunan). Notasi himpunan diatas menyeleksi semua bilangan bulat non-negatif dan lebih kecil dari 64 yang, jika dijadikan input untuk S_5, mengakibatkan persamaan antara kedua XOR terpenuhi.

Jika $NS_a(\alpha, \beta)$ tidak sama dengan 32, maka kita katakan bahwa ada korelasi antara input dan output S_a. Yang dicari adalah $NS_a(\alpha, \beta)$ yang memaksimalkan nilai $|NS_a(\alpha, \beta) - 32|$ (tabel untuk semua $NS_a(\alpha, \beta)$ dapat digunakan untuk mencari nilai maksimal). Sebagai contoh,

$$NS_5(16, 15) = 12$$

yang berarti bit input nomor 4 dari S_5 sama dengan XOR semua bit output S_5, dengan probabilitas $12/64 \approx 0.19$. Ini nilai optimal untuk $NS_a(\alpha, \beta)$. Dengan melibatkan ekspansi E dan permutasi P dalam fungsi *cipher f* DES, kita dapatkan persamaan

$$X[15] \oplus F(X, K)[7, 18, 24, 29] = K[22] \tag{8.3}$$

dengan probabilitas 0.19 untuk suatu kunci K yang ditetapkan dan suatu X yang terpilih secara acak. Karena $NS_a(16, 15)$ merupakan nilai optimal, persamaan 8.3 merupakan persamaan linear dengan probabilitas terbaik untuk S_5, jadi merupakan persamaan yang optimal.

8.2.2 Perkiraan Linear untuk DES

Setelah mendapatkan perkiraan linear yang baik untuk S-boxes, sekarang kita harus mencari perkiraan linear untuk algoritma DES secara keseluruhan. Sebagai contoh, kita gunakan algoritma DES dengan 3 putaran. Dengan mengaplikasikan persamaan 8.3 pada putaran pertama, kita dapatkan:

$$X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22] \tag{8.4}$$

dengan probabilitas 12/64, dimana
X_2 merupakan input putaran kedua,
K_1 merupakan kunci putaran pertama,
P_L merupakan 32 bit pertama (*low 32 bits*) dari P, dan
P_H merupakan 32 bit terahir(*high 32 bits*) dari P.
Demikian juga pada putaran terahir, kita dapatkan:

$$X_2[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_3[22] \tag{8.5}$$

juga dengan probabilitas 12/64. Kita kombinasikan kedua persamaan diatas untuk mendapatkan:

$$P_H[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus P_L[15] \oplus C_L[15] = K_1[22] \oplus K_3[22]. \tag{8.6}$$

Probabilitas bahwa persamaan 8.6 berlaku untuk sembarang naskah asli P dan naskah acaknya C adalah

$$(12/64)^2 + (1 - 12/64)^2 \approx 0.70.$$

Karena persamaan 8.3 merupakan perkiraan linear terbaik untuk fungsi *cipher* F, persamaan 8.6 merupakan persamaan terbaik untuk DES 3 putaran. Kita dapat gunakan algoritma 1 untuk mencari $K_1[22] \oplus K_3[22]$.

Seberapakah tingkat kesuksesan algoritma 1? Jika $|p - 1/2|$ cukup kecil, maka tingkat kesuksesan algoritma 1 dapat dirumuskan sebagai berikut:

$$\int_{-2\sqrt{N}|p-1/2|}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

Mari kita lihat bagaimana kita dapatkan rumus diatas. Algoritma 1 dapat dipandang sebagai eksperimen statistika dengan distribusi binomial:

- jumlah percobaan N adalah jumlah naskah asli yang digunakan, dan
- kita harapkan bahwa Np percobaan mematuhi persamaan 8.1.

Kita umpamakan bahwa $p > 0.5$ (untuk $p < 0.5$ analisanya sangat mirip karena simetris). Kita dapatkan

$$\begin{aligned} \mu &= Np \\ \sigma^2 &= Np(1 - p). \end{aligned}$$

Algoritma 1 (dengan $p > 0.5$) dianggap sukses jika mayoritas naskah asli yang digunakan mematuhi persamaan 8.1, dengan kata lain jumlah naskah asli yang mematuhi persamaan 8.1 melebihi $N/2$. Tingkat kesuksesan algoritma 1 adalah rasio eksperimen yang sukses dari semua kemungkinan eksperimen. Jadi kita harus melihat distribusi μ untuk semua kemungkinan eksperimen (distribusi ini istilahnya *sampling distribution*). Menurut teori probabilitas, *sampling distribution* untuk kasus ini adalah distribusi normal dengan deviasi standard

$$\begin{aligned} \frac{\sigma}{\sqrt{N}} &= \frac{\sqrt{Np(1-p)}}{\sqrt{N}} \\ &= \sqrt{p(1-p)} \\ &\approx 1/2 \end{aligned}$$

untuk nilai $|p - 1/2|$ yang kecil. Kurva distribusi untuk kasus ini berbentuk lonceng, dan untuk keperluan integral, kita dapat menggeser kurva lonceng sehingga μ terletak pada titik nol. Titik representasi untuk $N/2$ harus dibagi dengan $\sqrt{N}$ dan digeser menjadi

$$\frac{N/2 - Np}{\sqrt{N}} = \sqrt{N}(1/2 - p).$$

Supaya integral dapat menggunakan rumus distribusi normal standard dengan deviasi standard 1, titik representasi $N/2$ harus dibagi lagi dengan deviasi standard diatas menjadi:

$$\begin{aligned} \frac{\sqrt{N}(1/2-p)}{1/2} &= 2\sqrt{N}(1/2-p) \\ &= -2\sqrt{N}|p-1/2|. \end{aligned}$$

Kurva lonceng standard dengan deviasi standard 1, μ terletak pada titik 0, dan representasi untuk $N/2$ terletak pada titik $-2\sqrt{N}|p-1/2|$, merepresentasikan distribusi hasil eksperimen untuk semua kemungkinan eksperimen yang menggunakan N naskah asli. Jadi representasi hasil eksperimen yang sukses (dan hanya eksperimen yang sukses) akan terletak disebelah kanan titik representasi untuk $N/2$, dan tingkat kesuksesan algoritma 1 adalah area dibawah kurva lonceng mulai dari titik representasi untuk $N/2$ kekanan. Oleh sebab itu integral dimulai dari titik $-2\sqrt{N}|p-1/2|$.

N	$\frac{1}{4}\|p-1/2\|^{-2}$	$\frac{1}{2}\|p-1/2\|^{-2}$	$\|p-1/2\|^{-2}$	$2\|p-1/2\|^{-2}$
Sukses	84.1%	92.1%	97,7%	99.8%

Tabel 8.5: Tingkat kesuksesan algoritma 1

Tabel 8.5 menunjukkan tingkat kesuksesan algoritma 1. Untuk DES 3 putaran menggunakan persamaan 8.6 sebagai perkiraan, tingkat kesuksesan 97.7% membutuhkan

$$\begin{aligned} N &= |p-1/2|^{-2} \\ &= |0.7-0.5|^{-2} \\ &= 25 \end{aligned}$$

naskah asli.

Untuk DES 5 putaran, kita aplikasikan persamaan 8.3 pada putaran kedua dan keempat. Kita aplikasikan persamaan dibawah ini (yang mempunyai probabilitas 22/64)

$$X[27,28,30,31] \oplus F(X,K)[15] = K[42,43,45,46] \tag{8.7}$$

pada putaran pertama dan terahir. Kita dapat gabungkan keempat persamaan untuk menghasilkan

$$\begin{aligned} & P_H[15] \oplus P_L[7,18,24,27,28,29,30,31] \\ & \oplus C_H[15] \oplus C_L[7,18,24,27,28,29,30,31] \\ & = K_1[42,43,45,46] \oplus K_2[22] \oplus K_4[22] \oplus K_5[42,43,45,46]. \end{aligned} \tag{8.8}$$

Bagaimana kita mencari probabilitas bahwa persamaan 8.8 berlaku untuk sembarang naskah asli P dengan naskah acaknya C? Kita gunakan rumus probabilitas penggabungan variabel acak biner yang independen

$$X_1 \oplus X_2 \oplus \ldots \oplus X_n = 0$$

dimana setiap X_i mempunyai probabilitas p_i untuk menjadi 0 (jadi mempunyai probabilitas $(1 - p_i)$ untuk menjadi 1). Rumusnya adalah sebagai berikut:

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2).$$

Jadi untuk penggabungan empat persamaan diatas, kita dapatkan probabilitas

$$\begin{aligned} 1/2 + 2^{4-1} \prod_{i=1}^{4} (p_i - 1/2) &= 1/2 + 2^3(12/64 - 32/64)^2(22/64 - 32/64)^2 \\ &= 0.519 \end{aligned}$$

bahwa persamaan 8.8 berlaku untuk sembarang naskah asli P dengan naskah acaknya C. Untuk DES 5 putaran menggunakan persamaan 8.8 sebagai perkiraan, tingkat kesuksesan 97.7% membutuhkan

$$\begin{aligned} N &= |p - 1/2|^{-2} \\ &= |0.519 - 0.5|^{-2} \\ &= 2770 \end{aligned}$$

naskah asli.

Dalam papernya [mat93], Mitsuru Matsui memberikan tabel persamaan terbaik untuk DES dengan berbagai jumlah putaran. Untuk DES 16 putaran, diperlukan $|1.49 \times 2^{-24}|^{-2} \approx 2^{47}$ naskah asli untuk mencari 2 bit kunci. Di bagian berikut akan ditunjukkan cara untuk mendapatkan beberapa bit kunci sekaligus.

8.2.3 Known Plaintext Attack DES

Beberapa bit sekaligus dapat ditemukan untuk DES N putaran menggunakan algoritma 2 dengan persamaan paling efektif untuk $N - 1$ putaran. Sebagai contoh, untuk DES 8 putaran, kita gunakan persamaan paling efektif DES 7 putaran dikombinasikan dengan fungsi *cipher* F menjadi:

$$\begin{gathered} P_H[7, 18, 24] \oplus P_L[12, 16] \oplus C_H[15] \oplus C_L[7, 18, 24, 29] \oplus F_8(C_L, K_8)[15] \\ = K_1[19, 23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22]. \end{gathered} \tag{8.9}$$

Meskipun K_8 terdiri dari 48 bit, jumlah bit K_8 yang mempengaruhi hasil dari $F_8(C_L, K_8)[15]$ hanya ada 6, yaitu bit 42 sampai dengan 47, jadi diperlukan $2^6 = 64$ *counters* (yang merepresentasikan 64 kandidat kunci parsial 6 bit) untuk algoritma 2.

Analisa tingkat kesuksesan algoritma 2 agak sulit dan tidak akan dibahas disini. Pembaca yang ingin mendalami analisa tersebut dipersilahkan untuk membaca [mat93] yang memuat tabel 8.6 sebagai tingkat kesuksesan algoritma 2.

N	$2\|p-1/2\|^{-2}$	$4\|p-1/2\|^{-2}$	$8\|p-1/2\|^{-2}$	$16\|p-1/2\|^{-2}$
Sukses	48.6%	78.5%	96.7%	99.9%

Tabel 8.6: Tingkat kesuksesan algoritma 2

Jadi untuk tingkat kesuksesan 96.7% diperlukan sekitar $8|1.95 \times 2^{-10}|^{-2} \approx 2^{21}$ naskah asli. Karena sifat simetris yang terdapat pada putaran proses enkripsi DES, terdapat 2 persamaan paling efektif yang dapat digunakan untuk mendapatkan 12 bit kunci. Ditambah dengan 2 bit kunci yang didapatkan menggunakan algoritma 1, total 14 bit kunci didapatkan dari *linear cryptanalysis*. Sisanya, $56 - 14 = 42$ bit kunci dapat dicari dengan cara *exhaustive search*.

Untuk DES 16 putaran penuh, diperlukan $8|1.19 \times 2^{-22}|^{-2} \approx 2^{47}$ naskah asli menggunakan persamaan:

$$\begin{aligned} P_H[7,18,24] \oplus P_L[12,16] \oplus C_H[15] \oplus C_L[7,18,24,29] \oplus F_{16}(C_L, K_{16})[15] \\ = K_1[19,23] \oplus K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \oplus K_8[44] \\ \oplus K_9[22] \oplus K_{11}[22] \oplus K_{12}[44] \oplus K_{13}[22] \oplus K_{15}[22]. \end{aligned} \tag{8.10}$$

8.3 Pelajaran dari Cryptanalysis DES

Meskipun secara teoritis *linear cryptanalysis* menunjukkan sedikit kelemahan DES, secara umum dan praktis DES cukup tahan terhadap *differential cryptanalysis* dan *linear cryptanalysis*. Ini membantu menghilangkan kecurigaan bahwa NSA sengaja membuat DES lemah agar mereka dapat memecahkannya. (Tetapi dengan kemajuan teknologi komputer sekarang DES rentan terhadap serangan *brute force search*.) Sebaliknya, berbagai macam *block cipher* yang dirancang sebelum DES ternyata sangat rentan terhadap *differential* dan *linear cryptanalysis*. Pelajaran ini diterapkan dalam merancang *block cipher* modern yang mewajibkan *strict avalanche criterion* untuk setiap fungsi dalam enkripsi dimana perubahan 1 bit input menyebabkan setiap bit output mempunyai probabilitas 0.5 untuk berubah, independen dari bit lainnya.

8.4 Ringkasan

Dalam bab ini kita telah bahas dua teknik *cryptanalysis*. Teknik pertama untuk mencari kecenderungan dalam fungsi enkripsi adalah *differential cryptanalysis*. Teknik kedua adalah *linear cryptanalysis*. Perancangan *block cipher* kini memperhatikan ketahanan terhadap dua teknik *cryptanalysis* tersebut.