KEMINFO UTS

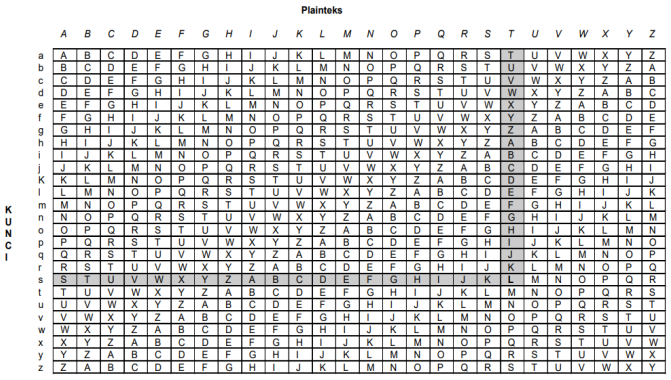
Kriptografi klasik vigenere

Vigenere Chipher → menggunakan **Bujursangkar Vigènere** untuk melakukan **enkripsi.**

Caranya tinggal nyocokin plain teks sama key per huruf. Contoh:
plain teks = thisplaintext
key = sony

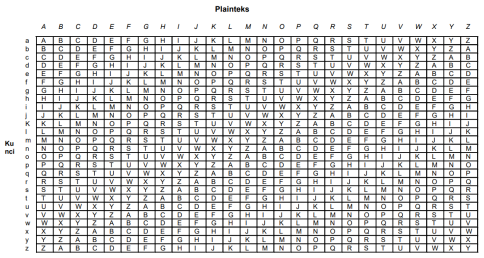
cara enkripsinya:
sesuain dulu panjang kunci dengan plainteks
plain teks = thisplaintext → 13 huruf
key = sonysonysonys → 13 huruf juga

terus tinggal cocokin deh, begini contohnya:



Gambar 4.3 Enkripsi huruf T dengan kunci S

si **huruf T dienkripsi** dengan **kunci s** hasilnya jadi **huruf L**, gitu seterusnya masa gangerti.



Gambar 4.2 Bujursangkar *Vigènere*

Kalo di sebelah kan pake tabel, tapi kalo lupa tabelnya tinggal pake rumusnya:
$c_i(p) = (p + k_i) \bmod 26$

Kunci: $K = k_1, k_2 \ldots k_m$
k_i untuk $1 \le i \le m$ menyatakan jumlah pergeseran pada huruf ke-i.

gampangnya tinggal geser indeks plainteks sejauh indeks key, trus karena abjad cuma sampe 26, jadi di modulo-in 26. Indeks dimulai dari 0.

Contoh soal:
Plaintext = institut pertanian
Key = key

Plain = institut pertanian
Key = keykeyke ykeykeyke

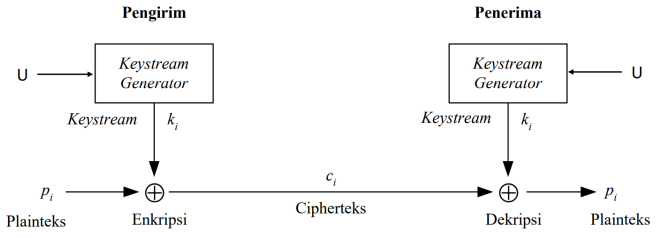
Huruf ke: (dari index 0)
k = 10, e = 4, y = 24
Geser karakter sejauh kunci atau
Gunakan rumus: **(plaintext + key) % 26**

i = (8 + 10) % 26 = 18 → **s**
n = (13 + 4) % 6 → **f**

dst.

Kriptografi sandi alir LFSR

Chiper/sandi alir → **Mengenkripsi** plainteks menjadi chiperteks setiap **bit per bit** dengan bit-bit kunci.



Gambar 1 Diagram *cipher* alir

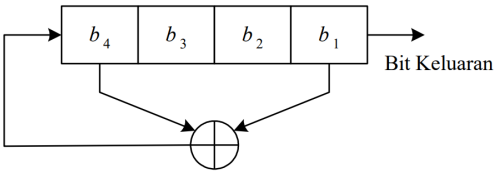
Cara enkripsi/dekripsinya tinggal di XOR (plain text sama keystreamnya bit per bit):
$c_i = p_i \oplus k_i$

Contoh:

pada skema sandi alir, yang biar pesannya aman itu **tergantung keystreamnya**. Semakin acak dan ga ketebak keystreamnya semakin aman.

buat nge-generate keystream, ada yang namanya keystream generator. Salah satunya LSFR atau Linear Feedback Shift Register. Simplenya, **geser bit paling kanan** sampe keluar, trus **ganti bit paling kir**i pake formula dibawah:

- Contoh LFSR 4-bit



- Fungsi umpan balik:

$$b_4 = f(b_1, b_4) = b_1 \oplus b_4$$

Plainteks: 1100101010100110001

Keystream: 1000110000101001101 ⊕ — Enkripsi

Cipherteks: 0100011010001111100

Keystream: 1000110000101001101 ⊕ — Dekripsi

Plainteks: 1100101010100110001

- Contoh: jika LFSR 4-bit diinisialisasi dengan U = 1111

i	Isi Register	Bit Keluaran
0	1 1 1 1	
1	0 1 1 1	1
2	1 0 1 1	1
3	0 1 0 1	1
4	1 0 1 0	1
5	1 1 0 1	0
6	0 1 1 0	1
7	0 0 1 1	0
8	1 0 0 1	1
9	0 1 0 0	1
10	0 0 1 0	0
11	0 0 0 1	0
12	1 0 0 0	1
13	1 1 0 0	0
14	1 1 1 0	0

- Barisan bit acak: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 ...
- Periode LFSR n-bit: $2^n - 1$

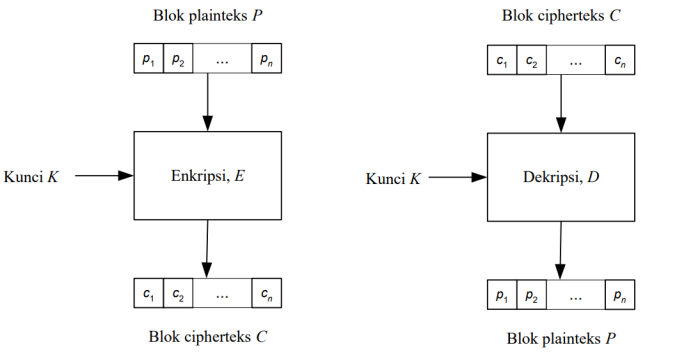
tapi kelemahannya, keystream pake LSFR bakal **berulang setiap 2^n-1 sekali**.

Kriptografi sandi blok

sandli blok → mode enkripsi/deskripsi suatu pesan **blok per blok** dengan panjang yang sama

Blok plainteks berukuran *n* bit:
$P = (p_1, p_2, ..., p_n)$, $p_i \in \{0, 1\}$

Blok cipherteks berukuran *n* bit:
$C = (c_1, c_2, ..., c_n)$, $c_i \in \{0, 1\}$



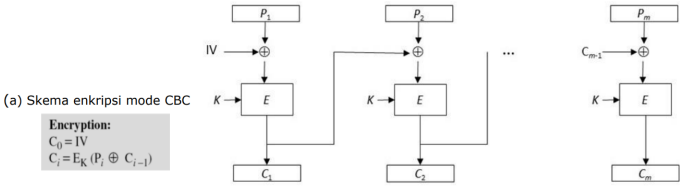
Contoh algoritme block:

- *Daftar block cipher:*

1. Lucifer
2. DES
3. GOST
4. 3DES (Triple-DES)
5. RC2
6. RC5
7. Blowfish
8. AES
9. IDEA
10. LOKI
11. FEAL
12. CAST-128
13. CRAB
14. SAFER
15. Twofish
16. Serpent
17. RC6
18. Camellia
19. 3-WAY
20. MMB
21. Skipjack
22. TEA
23. XTEA
24. SEED
25. Coconut
26. Cobra
27. MARS
28. BATON
29. CRYPTON
30. LEA, dll

2. Cipher Block Chaining

→ mode enkripsi blok secara **sequential**, ada dependensi antar bloknya.



Ciperteks sebelumnya dipake lagi buat nge-enkripsi plainteks saat ini (di-XOR sama plainteks). Khusus buat blok pertama, XORnya sama **Initialization Vector (IV)**. Contoh:

1010001000111010 1001

Bagi plainteks menjadi blok-blok yang berukuran 4 bit:

1010 0010 0011 1010 1001

atau dalam notasi HEX adalah A23A9.

Misalkan kunci (K) yang digunakan adalah (panjangnya juga 4 bit)

1011

atau dalam notasi HEX adalah B. Sedangkan IV yang digunakan seluruhnya bit 0 (Jadi, C_0 = 0000)

Fungsi enkripsi E yang digunakan sama seperti sebelumnya: XOR-kan blok plainteks P_i dengan K, kemudian geser secara *wrapping* bit-bit dari $P_i \oplus K$ satu posisi ke kiri.

$$E_K(P) = (P \oplus K) << 1$$

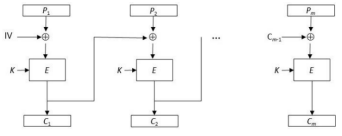
Cara ngerjainnya:

C_1 diperoleh sebagai berikut:
$P_1 \oplus C_0 = 1010 \oplus 0000 = 1010$

Enkripsikan hasil ini dengan fungsi E sbb:
$1010 \oplus K = 1010 \oplus 1011 = 0001$
Geser (*wrapping*) hasil ini satu bit ke kiri: 0010
Jadi, C_1 = 0010 (atau 2 dalam HEX)

C_2 diperoleh sebagai berikut:
$P_2 \oplus C_1 = 0010 \oplus 0010 = 0000$
$0000 \oplus K = 0000 \oplus 1011 = 1011$
Geser (*wrapping*) hasil ini satu bit ke kiri: 0111
Jadi, C_2 = 0111 (atau 7 dalam HEX)

C_3 diperoleh sebagai berikut:
$P_3 \oplus C_2 = 0011 \oplus 0111 = 0100$
$0100 \oplus K = 0100 \oplus 1011 = 1111$
Geser (*wrapping*) hasil ini satu bit ke kiri: 1111
Jadi, C_3 = 1111 (atau F dalam HEX)



Ada 5 mode operasi pada cipher blok:

1. Electronic Code Book (ECB)

→ setiap blok dienkripsi/dekripsi **secara independen**, ga mentingin blok sebelum/sesudahnya. Contoh:

$$E_K(P) = (P \oplus K) << 1$$

Enkripsi:

	1010	0010	0011	1010	1001
	1011	1011	1011	1011	1011 ⊕
Hasil *XOR*:	0001	1001	1000	0001	0010
Geser 1 bit ke kiri:	0010	0011	0001	0010	0100
Dalam notasi HEX:	2	3	1	2	4

Jadi, hasil enkripsi plainteks

1010001000111010100 (A23A9 dalam notasi HEX)

adalah

0010001100010010100 (23124 dalam notasi HEX)

Karakteristik: tiap blok dengan isi yang sama bakal di enkripsi ke cipher yang sama juga (liat blok 1010 di contoh atas),

	1010	0010	0011	1010	1001
	1011	1011	1011	1011	1011 ⊕
Hasil *XOR*:	0001	1001	1000	0001	0010
Geser 1 bit ke kiri:	0010	0011	0001	0010	0100
Dalam notasi HEX:	2	3	1	2	4

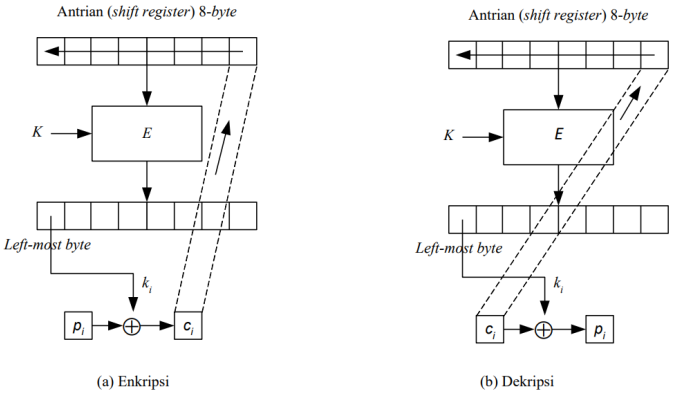
makanya biasanya dibikin "code book" buat tiap kemungkinan cipher teks yang berkoresponden sama plainteksnya.

Plainteks	Cipherteks
0000	0100
0001	1001
0010	1010
...	...
1111	1010

3. Cipher-feedback (CFB)

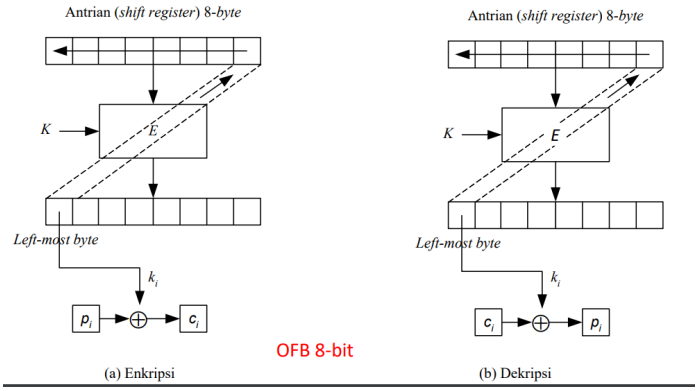
→ Mengatasi kekurangan pada mode CBC apabila diterapkan pada pengiriman data yang belum mencapai ukuran satu blok.

Mode CFB-8 bit untuk ukuran blok 64 bit (8 byte)



4. Output-feedback (OFB)

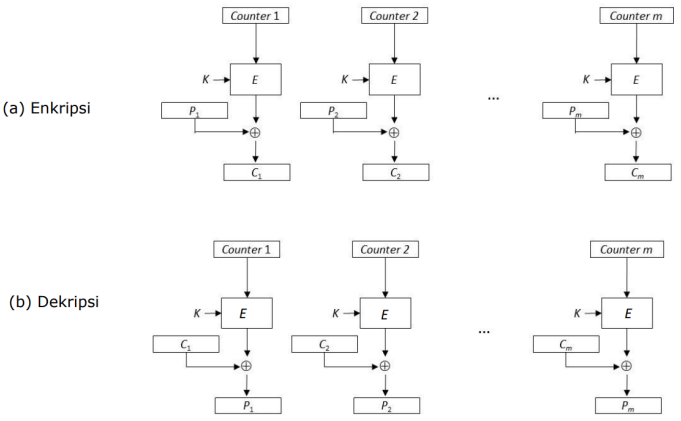
→ mirip dengan CFB, beda naro paling kanannya aja



OFB 8-bit

5. Counter Mode

→ Pada mulanya, untuk enkripsi blok pertama, counter diinisialisasi dengan sebuah nilai. Selanjutnya, untuk enkripsi blok-blok berikutnya counter dinaikkan (increment) nilainya satu (counter = counter + 1).



Kriptografi kunci publik RSA

1. p dan q bilangan prima (rahasia)
2. $n = p \cdot q$ (tidak rahasia)
3. $\phi(n) = (p-1)(q-1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia)

   Syarat: PBB(e, $\phi(n)$) = 1 , PBB = pembagi bersama terbesar = *gcd*
5. d (kunci dekripsi) (rahasia)

   d dihitung dari $d \equiv e^{-1} \bmod (\phi(n))$
6. m (plainteks) (rahasia)
7. c (cipherteks) (tidak rahasia)

Prosedur Pembangkitan Sepasang Kunci

1. Pilih dua bilangan prima, p dan q (sebaiknya $p \neq q$)
2. Hitung $n = pq$.
3. Hitung $\phi(n) = (p-1)(q-1)$.
4. Pilih sebuah bilangan bulat e sebagai kunci publik, e harus relatif prima terhadap $\phi(n)$.
5. Hitung kunci dekripsi, d, dengan persamaaan

   $ed \equiv 1$ (**mod** $\phi(n)$) atau $d \equiv e^{-1}$ (mod ($\phi(n)$)

Hasil dari algoritma di atas:
- Kunci publik adalah pasangan (e, n)
- Kunci privat adalah pasangan (d, n)

Enkripsi

1. Jika pesan berukuran besar, nyatakan pesan menjadi blok-blok plainteks yang lebih kecil: m_1, m_2, m_3, ...

   (syarat: $0 \leq m_i < n-1$)

2. Hitung cipherteks c_i untuk plainteks m_i menggunakan kunci publik e dengan persamaan

   $c_i = m_i^e$ **mod** n

Dekripsi

1. Misalkan cipherteks adalah c_1, c_2, c_3, ...

2. Hitung kembali blok plainteks m_i dari blok cipherteks c_i menggunakan kunci privat d dengan persamaan

   $m_i = c_i^d$ **mod** n,

Contoh:

1. Pilih dua buah bilangan prima, p dan q
   Misalnya :
   p = 3
   q = 11
2. Carilah nilai n
   n = p x q
   n = 3 x 11
   n = 33
3. Carilah nilai m
   m = (p-1) x (q-1)
   m = (3-1) x (11-1)
   m = 20

4. Mencari e yang relatif prima terhadap m,
   Rumus Algoritma Euclidean :

$$r_0 = q_1 r_1 + r_2, \quad 0 < r_2 < r_1$$
$$r_1 = q_2 r_2 + r_3, \quad 0 < r_3 < r_2$$
$$r_n = \cdots$$

Percobaan 1 : nilai e = 2, m = 20
gcd(e,m) = 1
gcd(2,20) = 1
$r_0 = 2,\ r_1 = 20$
$r_0 = q_1 . r_1 + r_2$
2 = 0.20 + 2 Keterangan : nilai awal q = 0,
$r_2 = r_0 - (q_1 . r_1)$
$r_2 = 2 - (0.20)$
$r_2 = 2$

Karena nilai r belum 0 maka dilanjutkan :
$r_1 = q_2 . r_2 + r_3$
20=10.2 + 0 Keterangan : $q_2 = r_1 : r_2$
$q_2 = 20:2$
$q_2 = 10$

Percobaan 2 : nilai e = 7, m = 20
gcd(e,m) = 1
gcd(7,20) = 1
$r_0 = 7,\ r_1 = 20$
$r_0 = q_1 . r_1 + r_2$
7 = 0.20+ 7 Ket : nilai awal q = 0,
$r_2 = r_0 - (q_1 . r_1)$
$r_2 = 7 - (0.20)$
$r_2 = 7$

Karena nilai r belum 0 maka dilanjutkan:
$r_1 = q_2 . r_2 + r_3$
20= 2.7 + 6 Ket : $q_2 = r_1 : r_2$
$q_2 = 20:7$
$q_2 = 2$
$r_3 = r_1 - (q_2 . r_2)$
$r_3 = 20 - (2.7)$
$r_3 = 6$

Karena nilai r belum 0 maka dilanjutkan:
$r_2 = q_3 . r_3 + r_4$
7 = 1.6 + 1 Ket : $q_3 = r_2 : r_3$
$q_3 = 7:6$
$q_3 = 1$
$r_4 = r_2 - (q_3 . r_3)$
$r_4 = 7 - (1.6)$
$r_4 = 1$

Karena nilai r belum 0 maka dilanjutkan:
$r_3 = q_4 . r_4 + r_5$
6 = 1.6 + 0 Ket : $q_4 = r_3 : r_4$
$q_4 = 6:6$
$q_4 = 1$
$r_5 = r_3 - (q_4 . r_4)$
$r_5 = 6 - (1.6)$
$r_5 = 0$

```
e = 7
m = 20
      e x d mod m  = 1
      7 x 3 mod 20 = 1
```

Keterangan : Nilai d adalah nilai yang kita tebak dan coba cocokkan, karena 7 x 3 = 21 dan jika 21 mod 20 maka hasilnya 1, maka 3 adalah angka yang tepat untuk niali d.

6 . Dengan didapat nilai d yaitu 3, maka pasangan kunci adalah :

- Kunci Public = (e,n) yaitu (7,33)
- Kunci Private = (d,n) yaitu (3,33)

ENKRIPSI

Enkripsi, mengubah pesan asli menjadi sandi adalah inti dari kriptografi. Oleh karena itu, tujuan dari ekspansi kunci adalah untuk dapat mengenkripsi pesan dengan kunci public yang telah didapat. Berikut adalah rumus dan proses enkripsi :

Misal pesan yang ingin kita enkripsi adalah 14, angka ini hanya contoh, jika pesan berupa huruf maka conversikan ke dalam bentuk angka dengan ASCII code. Kemudian enkripsi dengan rumus :

$$C = M^e \, (mod n)$$

Keterangan : C = Cipherteks, M = Message (Pesan), maka :

$$C = 14^7 \, (mod 33) \, , C = 20$$

DEKRISPI

Untuk mengembalikan pesan asli, kita ambil pesan sandi yang tadi sudah didapat yaitu 20, maka dekripsikan dengan rumus berikut :

$$M = C^d \, (mod n)$$

Keterangan : C = Cipherteks, M = Message (Pesan), maka :

$$M = 20^3 \, (mod 33) \, , C = 14$$

Kriptografi fungsi Hash

fungsi hash → Fungsi yang **mengkompresi pesan (M)** berukuran sembarang **menjadi string (h)** yang berukuran **fixed**. BUKAN FUNGSI ENKRIPSI.

Irreversible (tidak bisa dikembalikan menjadi pesan semula)

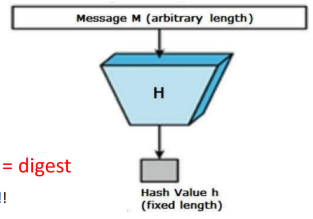
Fungsi Hash:

***h* = *H*(*M*)**

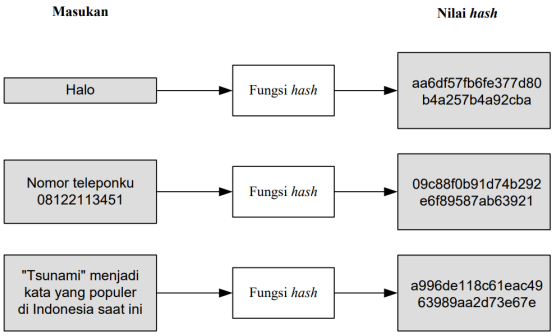
h<<<<M

h = Hash value = message digest = digest

Contoh: *size*(*M*) = 1 MB → *size*(*h*) = 256 bit !!!!



Contoh:



Manfaat:

1. Menjaga integritas data

→ kalo pesan diubah, meskipun dikit doang, maka hasil hashnya berubah banget.

(ii) Misal 33 diubah menjadi 32

```
Pada bulan Oktober 2004 ini, suhu udara kota Bandung terasa
lebih panas dari hari-hari biasanya. Menurut laporan Dinas
Meteorologi Kota Bandung, suhu tertinggi kota Bandung
adalah 32 derajat Celcius pada Hari Rabu, 17 Oktober yang
lalu. Suhu terseut sudah menyamai suhu kota Jakarta pada
hari-hari biasa. Menurut Kepala Dinas Meteorologi,
peningkatan suhu tersebut terjadi karena posisi bumi
sekarang ini lebih dekat ke matahari daripada hari-hari
biasa.

Sebutan Bandung sebagai kota sejuk dan dingin mungkin tidak
lama lagi akan tinggal kenangan. Disamping karena faktor
alam, jumlah penduduk yang padat, polusi dari pabrik di
sekita Bandung, asap knalpot kendaraan, ikut menambah
kenaikan suhu udara kota.
```

Nilai MD5: **2D1436293FAEAF405C27A151C0491267**

Sebelum diubah : $MD5_1$ = **2F82D0C845121B953D57E4C3C5E91E63**
Sesudah diubah : $MD5_2$ = **2D1436293FAEAF405C27A151C0491267**
Verifikasi: $MD5_1 \neq MD5_2$ (arsip sudah diubah)

2. Menghemat waktu pengiriman

→ Ketimbang mengirim salinan arsip tersebut secara keseluruhan ke komputer pusat (yang membutuhkan waktu transmisi lama), lebih mangkus mengirimkan message digest-nya.

3. Menormalkan panjang data yang beraneka ragam

→ - Untuk menyeragamkan panjang field password di

Sifat:

Sifat-sifat fungsi *hash H*:

a) collision resistance : sangat sukar menemukan dua input *a* dan *b* sedemikian sehingga $H(a) = H(b)$

b) preimage resistance: untuk sembarang output *y*, sukar menemukan input *a* sedemikian sehingga $H(a) = y$

c) second preimage resistance – untuk input *a* dan output $y = H(a)$, sukar menemukan input kedua *b* sedemikian sehingga $H(b) = y$

dalam basisdata, password disimpan dalam bentuk nilai hash (panjang nilai hash tetap).

Kolisi

- Kolisi (*collision*) adalah kondisi dua *string* sembarang memiliki nilai *hash* yang sama.
- Adanya kolisi menunjukkan fungsi *hash* tidak aman secara kriptografis

Kriptografi simetrik DES

Proses algoritma DES dapat dilakukan dengan langkah-langkah berikut :

- Proses awal dimulai dengan blok plain text 64-bit diproses dengan fungsi Initial Permutation (IP).
- Initial Permutation (IP) kemudian dilakukan pada plain text.
- Selanjutnya, Initial Permutation (IP) membuat dua bagian dari blok permutasi yang disebut sebagai Left Plain Text (LPT) dan Right Plain Text (RPT).
- Setiap LPT dan RPT melalui 16 putaran proses enkripsi.
- Akhirnya, LPT dan RPT bergabung kembali, dan Final Permutation (FP) dilakukan pada blok yang baru digabungkan.
- Hasil dari proses ini menghasilkan teks sandi 64-bit yang diinginkan.

Langkah proses enkripsi (langkah 4, di atas) bisa dipecah menjadi lima tahap yaitu :

- Transformasi kunci
- Ekspansi permutasi
- Permutasi S-Box
- Permutasi P-Box
- XOR dan swap

Contoh soal:
http://octarapribadi.blogspot.com/2012/10/contoh-enkripsi-dengan-algoritma-des.html