
UMSU PRESS
KRIPTOGRAFI
Dan Sekuriti Sistem
Ferdy Riza, Muhammad Haris, Farid Akbar Siregar,
Mahardika Abdi Prawira Tanjung, Andi Zulherry,
Zuli Agustina Gultom, Okvi Nugroho, Mhd Basri, Amrullah,
Mulkan Azhari, Fatma Sari Hutagalung, Hevlie Winda Nazry

Kriptografi Dan Sekuriti Sistem

Kriptografi Dan Sekuriti Sistem

Oleh:
Ferdy Riza, Muhammad Haris, Farid Akbar Siregar, Mahardika Abdi Prawira Tanjung, Andi Zulherry, Zuli Agustina Gultom, Okvi Nugroho, Mhd Basri, Amrullah, Mulkan Azhari, Fatma Sari Hutagalung, Hevlie Winda Nazry

Editor:
Dr. Al-Khowarizmi, S.Kom., M.Kom.

UMSU PRESS

Judul
Kriptografi Dan Sekuriti Sistem

Penulis
Ferdy Riza, Muhammad Haris, Farid Akbar Siregar, Mahardika Abdi Prawira Tanjung, Andi Zulherry, Zuli Agustina Gultom, Okvi Nugroho, Mhd Basri, Amrullah, Mulkan Azhari, Fatma Sari Hutagalung, Hevlie Winda Nazry

Editor
Dr. Al-Khowarizmi, S.Kom., M.Kom.

Layouter
Sartika Tri Anjayani, S.Kom.

Cetakan Pertama; Januari 2025
xiv + 204 hlm ; 15.5 x 23 cm

ISBN : 978-623-408-895-3 (Cetak)
E-ISBN : 978-623-408-900-4 (*e-book*/PDF)

Penerbit

Redaksi
Jalan Kapten Mukhtar Basri No 3 Medan, 20238
Telepon, 061-6626296, Fax. 061-6638296
Email; umsupress@umsu.ac.id
Website; http://umsupress.umsu.ac.id/
Anggota IKAPI Sumut, No: 38/Anggota Luar Biasa/SUT/2020
Anggota APPTI, Nomor: 005.053.1.09.2018
Anggota APPTIMA, Nomor: 01/B/AnggotaAPPTIMA/2023

DAFTAR **ISI**

DAFTAR **GAMBAR**

DAFTAR **TABEL**

Kata **Pengantar**

Tujuan menulis buku Kriptografi dan Sekuriti Sistem ini disusun sebagai rujukan bagi pembaca yag sdang mempelajari ilmu kriptografi dan sekuriti sistem. Pada Buku kriptografi dan sekuriti sistem ini membahas topik utama yaitu Pengantar kriptografi.

Buku ini masih belum sempurna, sehingga perlu dikaji baik oleh pemakai buku ini. Oleh karena itu penyusun berharap agar para pemakai buku ini dapat memberikan sumbangan saran untuk perbaikan dari buku kriptografi dan sekuriti sistem ini. Semoga buku ini dapat bermanfaat bagi para pembaca yang sedang mempelajari kriptografi dan sekuriti sistem, serta dapat meningkatkan kemampuan pembaca dalam menguasai konsep dasar, peran utama, metode, proses, dan algoritma kriptografi dan sekuriti sistem.

Medan, 01 Oktober 2024

Penulis

1.
Pengantar **Kripto**

1.1. Terminologi

(a) Pengirim dan Penerima pesan

Seorang pengirim pesan (*sender*) ingin mengirim pesan kepada seorang penerima (*receiver*). Pengirim menginginkan pesan dapat dikirim secara aman, yaitu ia yakin bahwa pihak lain tidak dapat membaca isi pesan.

(b) Pesan, Plainteks, dan Cipherteks

Pesan adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Nama lain untuk pesan adalah **plainteks** (*plaintext*) atau teks-jelas (*cleartext*).

Pesan dapat berupa data atau informasi yang dikirim (melalui kurir, saluran komunikasi data, dsb) atau yang disimpan di dalam media perekaman (kertas, storage, dsb).

Agar pesan tidak dapat dimengerti maknanya oleh pihak lain, maka pesan disandikan ke bentuk lain. Bentuk pesan yang tersandi disebut **cipherteks** (*ciphertext*) atau **kriptogram** (**cryptogram**).

Cipherteks harus dapat ditransformasi kembali menjadi plainteks.

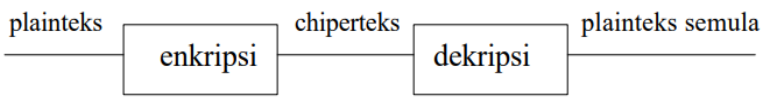
Contoh:

Plainteks : *uang disimpan di balik buku X*
Cipherteks : *j&kloP#d$gkh*7h^"tn%6^klp..t@*

(c) Enkripsi dan Dekripsi

Proses menyandikan plainteks menjadi cipherteks disebut enkripsi (*encryption*) atau *enciphering* (standard nama menurut ISO 7498-2).

Proses mengembalikan cipherteks menjadi plainteksnya disebut dekripsi (decryption) atau *deciphering* (standard nama menurut ISO 7498-2).



Gambar 1.1 Enkripsi dan Dekripsi

(d) Kriptografi

Kriptografi adalah ilmu sekaligus seni untuk menjaga keamanan pesan (*message*) [Schneier, 1996].

Praktisi (pengguna kriptografi) disebut **kriptografer** (*cryptographer*).

(e) Algoritma kriptografi dan Kunci

Algoritma kriptografi adalah:

- aturan untuk *enchipering* dan *dechipering*
- fungsi matematika yang digunakan untuk enkripsi dan dekripsi.

Kunci adalah parameter yang digunakan untuk transformasi *enciphering* dan *dechipering*.

(f) Sistem Kriptografi

Sistem kriptografi (atau ***cryptosystem*****)** adalah algoritma kriptografi, plainteks, cipherteks, dan kunci.

(g) Penyadap

Penyadap (*eavesdropper*) adalah orang yang mencoba menangkap pesan selama ditransmisikan.

Nama lain: *enemy*, *adversary*, *intruder*, *interceptor*, *bad guy*

(h) Kriptanalisis dan kriptologi

- **Kriptanalisis** (*cryptanalysis*) adalah ilmu dan seni untuk memecahkan chiperteks menjadi plainteks tanpa mengetahui *kunci* yang diberikan. Pelakunya disebut **kriptanalis**.
- **Kriptologi** (*cryptology*) adalah studi mengenai kriptografi dan kriptanalisis.

Persamaan kriptografer dan kriptanalis:

- Keduanya sama-sama menerjemahkan cipherteks menjadi plainteks

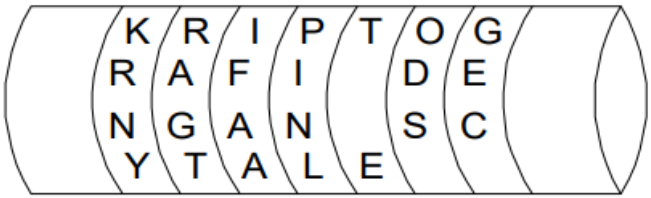
Perbedaan kriptografer dan kriptanalis:

- Kriptografer bekerja atas legitimasi pengirim atau penerima pesan
- Kriptanalis bekerja atas nama penyadap yang tidak berhak.

1.2. Sejarah kriptografi

Kriptografi sudah lama digunakan oleh tentara Sparta di Yunani pada permulaan tahun 400 SM. Mereka menggunakan alat yang namanya *scytale*.

Scytale: pita panjang dari daun *papyrus* + sebatang silinder Pesan ditulis horizontal (baris per baris). Bila pita dilepaskan, maka huruf-huruf di dalamnya telah tersusun membentuk pesan rahasia. Untuk membaca pesan, penerima melilitkan kembali silinder yang diameternya sama dengan diameter silinder pengirim.



Gambar 1.2 scytale

1.3. Aplikasi Kriptografi

Aplikasi kriptografi:

1. Pengiriman data melalui saluran komunikasi
2. Penyimpanan data di dalam *disk storage*.

Data ditransmisikan dalam bentuk cipherteks. Di tempat penerima cipherteks dikembalikan lagi menjadi plainteks. Data di dalam media penyimpanan komputer (seperti *hard disk*) disimpan dalam bentuk cipherteks. Untuk membacanya, hanya orang yang berhak yang dapat mengembalikan chiperteks menjadi plainteks.

Contoh-contoh enkripsi dan dekripsi pada data tersimpan:

1. Dokumen teks

Plainteks (plain.txt):

```
Ketika saya berjalan-jalan di pantai,
saya menemukan banyak sekali kepiting
yang merangkak menuju laut. Mereka
adalah anak-anak kepiting yang baru
menetas dari dalam pasir. Naluri
mereka mengatakan bahwa laut adalah
tempat kehidupan mereka.
```

Cipherteks (cipher.txt):

```
Ztâxzp/épêp/qtüyp{p}<yp{p}/sx/•p}âpx;
  épêp/|t}t|äzp}/qp}êpz/étzp{x/zt•x
}v êp }v/|tüp}vzpz/|t}äyä/{päâ=/\tütz
p psp{pw/p}pz<p}pz/zt•xâx}v/êp}
v/qpüä |t}tâpé/spüx/sp{p|/•péxü=/]
p{äüx |ttüzp/|t}vpâpzp}/qpwâp/{päâ
/psp{pw ât|•pâ/ztwxsä•p}/|tützp=
```

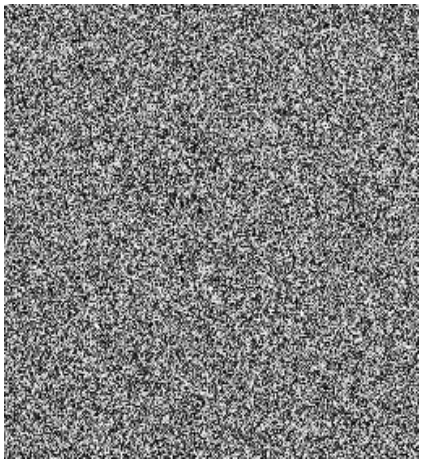
Hasil dekripsi terhadap berkas cipher.txt:

```
Ketika saya berjalan-jalan di pantai,
saya menemukan banyak sekali kepiting
yang merangkak menuju laut. Mereka
adalah anak-anak kepiting yang baru
menetas dari dalam pasir. Naluri
mereka mengatakan bahwa laut adalah
tempat kehidupan mereka.
```

2. Dokumen gambar

Plainteks (lena.bmp):

Cipherteks (lena2.bmp):



Hasil dekripsi terhadap berkas lena2.bmp menghasilkan gambar yang sama seperti lena.bmp.

3. Dokumen basisdata

Plainteks (siswa.dbf):

NIM	Nama	Tinggi	Berat
000001	Elin Jamilah	160	50
000002	Fariz RM	157	49
000003	Taufik Hidayat	176	65
000004	Siti Nurhaliza	172	67
000005	Oma Irama	171	60
000006	Aziz Burhan	181	54
000007	Santi Nursanti	167	59
000008	Cut Yanti	169	61
000009	Ina Sabarina	171	62

Cipherteks (siswa2.dbf):

NIM	Nama	Tinggi	Berat
000001	tüp}vzpz/|t}äyä/{ää	|äzp}	épêp
000002	|t}tâpé/spüx/	péxü=	ztwxsä•
000003	ât|•pâ/ztwxsä•p}	}/|tü	spüx/
000004	épêp/|t}t|äzp}/qpêpz	qp}êpz	wxsä
000005	étzp{x/zt•xâx}v êp}	päâ/psp	étzp{
000006	spüx/sp{p|/•péxü=/]	xâx}v	ttüzp/|
000007	Ztâxzp/épêp/qtüypp}<	äzp}	}äyä/{
000008	qpwåp/{päâ/psp{pw	Ztwxs	xâx}v
000009	}t|äzp}/qp}êpz/ép{	qp}êp	äzp}/qp

Keterangan: hanya *field* Nama, Berat, dan Tinggi yang dienkripsi. Hasil dekripsi terhadap berkas siswa2.dbf menghasilkan berkas yang sama seperti siswa.dbf.

Kehidupan saat ini dikelilingi oleh kriptografi, mulai:

- ATM tempat mengambil uang,
- Telepon genggam (HP),
- Komputer di lab/kantor,

- Internet,
- Gedung-gedung bisnis,
- sampai ke pangkalan militer

1.4. Kegunaan Kriptografi

Selain untuk menjaga kerahasiaan (confidentiality) pesan, kriptografi juga digunakan untuk menangani masalah keamanan yang mencakup dua hal berikut:

1. Keabsahan pengirim (*user authentication*).
   Hal ini berkaitan dengan keaslian pengirim. Dengan kata lain, masalah ini dapat diungkapkan sebagai pertanyaan: "Apakah pesan yang diterima benar-benar berasal dari pengirim yang sesungguhnya?"
2. Keaslian pesan (*message authentication*).
   Hal ini berkaitan dengan keutuhan pesan (*data integrity*). Dengan kata lain, masalah ini dapat diungkapkan sebagai pertanyaan: "Apakah pesan yang diterima tidak mengalami perubahan (modifikasi)?"
3. Anti-penyangkalan (*nonrepudiation*).
   Pengirim tidak dapat menyangkal (berbohong) bahwa dialah yang mengirim pesan.

1.5. Notasi Matematis

Misalkan:

C = chiperteks
P = plainteks dilambangkan

Fungsi enkripsi E memetakan P ke C,

$E(P) = C$

Fungsi dekripsi D memetakan C ke P,

$D(C) = P$

Karena proses enkripsi kemudian dekripsi mengembalikan pesan ke pesan asal, maka kesamaan berikut harus benar,

$$D(E(P)) = P$$

Kekuatan algoritma kriptografi diukur dari banyaknya kerja yang dibutuhkan untuk memecahkan data chiperteks menjadi plainteksnya. Kerja ini dapat diekivalenkan dengan waktu.

Semakin banyak usaha yang diperlukan, yang berarti juga semakin lama waktu yang dibutuhkan, maka semakin kuat algoritma kriptografinya, yang berarti semakin aman digunakan untuk menyandikan pesan.

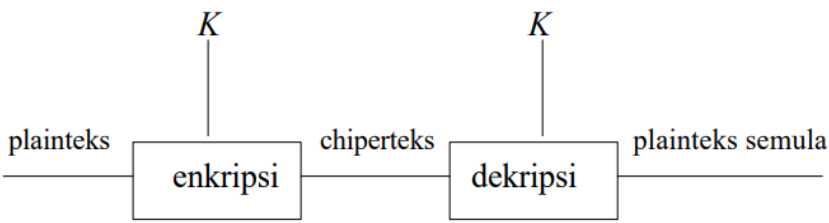
Jika kekuatan kriptografi ditentukan dengan menjaga kerahasiaan algoritmanya, maka algoritma kriptografinya dinamakan algoritma *restricted*. Algoritma *restricted* tidak cocok lagi saat ini.

Pada sistem kriptografi modern, kekuatan kriptografinya terletak pada kunci, yang berupa deretan karakter atau bilangan bulat, dijaga kerahasiaannya. Dengan menggunakan kunci K, maka fungsi enkripsi dan dekripsi menjadi

$$E_K(P) = C$$
$$D_K(C) = P$$

dan kedua fungsi ini memenuhi

$$D_K(E_K(P)) = P$$



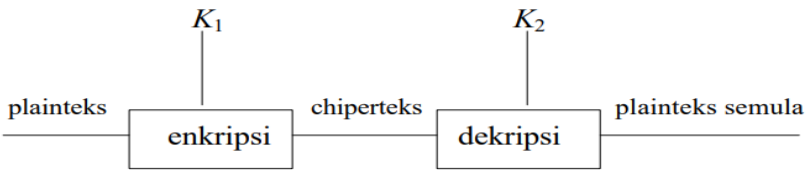
Gambar 1.3 Enkripsi dan dekripsi dengan kunci

Jika kunci enkripsi sama dengan kunci dekripsi, maka sistem kriptografinya disebut **sistem simetri** atau **sistem konvensional**. Algoritam kriptografinya disebut algoritma simetri atau algoritma konvensional .

Contoh algoritma simetri: *DES* (*Data Encyption Standard*).

Beberapa sistem kriptografi menggunakan kunci yang berbeda untuk enkripsi dan dekripsi. Misalkan kunci enkripsi adalah K_1 dan kunci dekripsi yang adalah K_2, yang dalam hal ini $K1 \neq K2$. Sistem kriptograsi semacam ini dinamakan sistem **sistem nirsimetri** atau **sistem kunci-publik**. Algoritam kriptografinya disebut algoritma nirsimetri atau algoritma kunci-publik.

Contoh algoritma nirsimetri: *RSA* (Rivest-Shamir-Adleman)



Gambar 1.4 Enkripsi dan dekripsi dengan kunci pada sistem nirsimetri

2.
Serangan (*Attack*) Teradap Kriptografi

2.1. Pendahuluan

Keseluruhan *point* dari kriptografi adalah menjaga kerahasiaan plainteks (atau kunci, atau keduanya) dari penyadap (*eavesdropper*) atau kriptanalis (*cryptanalyst*).

Penyadap bisa juga merangkap sebagai seorang kriptanalis. Nama lain penyadap:

- penyusup (*intruder*)
- penyerang (*attacker*)
- musuh (*enemy*, *adversaries*)
- pencegat (*interceptor*)
- lawan (opponent)

Penyadap berusaha mendapatkan data yang digunakan untuk kegiatan kriptanalisis (*cryptanalysis*). Kriptanalis berusaha mengungkap plainteks atau kunci dari data yang disadap. Kriptanalis juga dapat menemukan kelemahan dari sistem kriptografi yang pada akhirnya mengarah untuk menemukan kunci dan mengungkap plainteks.

2.2. Metode Penyadapan

Beberapa metode penyadapan data:

1. *Wiretapping*
   Penyadap mencegat data yang ditransmisikan pada saluran kabel komunikasi dengan menggunakan sambungan perangkat keras.
2. *Electromagnetic eavesdropping*

Penyadap mencegat data yang ditransmisikan melalui saluran *wireless*, misalnya radio dan *microwive*.

3. *Acoustic Eavesdropping*.
   Menangkap gelombang suara yang dihasilkan oleh suara manusia.

2.3. Metode Penyadapan

Yang dimaksud dengan serangan (*attack*) adalah setiap usaha (*attempt*) atau percobaan yang dilakukan oleh kriptanalis untuk menemukan kunci atau menemukan plainteks dari cipherteksnya. Secara umum, ada dua macam serangan:

1. *Exhaustive attack* atau *brute force attack*
   Percobaan yang dibuat untuk mengungkap plainteks atau kunci dengan mencoba semua kemungkinan kunci (*trial and error*).
   Asumsi yang digunakan:

a. Kriptanalis mengetahui algoritma kriptografi
b. Kriptanalis memiliki sebagian plainteks dan cipherteks yang bersesuaian.
   Caranya: plainteks yang diketahui dienkripsikan dengan setiap kemungkinan kunci, dan hasilnya dibandingkan dengan cipherteks yang bersesuaian.

Jika hanya cipherteks yang tersedia, cipherteks tersebut didekripsi dengan dengan setiap kemungkinan kunci dan plainteks hasilnya diperiksa apakah mengandung makna.

Misalkan sebuah sistem kriptografi membutuhkan kunci yang panjangnya 8 karakter, karakter dapat berupa angka (10 buah), huruf (26 huruf besar dan 26 huruf kecil), maka jumlah kunci yang harus dicoba adalah sebanyak

$$62 \times 62 \times 62 \times 62 \times 62 \times 62 \times 62 \times 62 = 62^8$$

Buah.

Secara teori, serangan secara *exhaustive* ini dipastikan berhasil mengungkap plainteks tetapi dalam waktu yang sangat lama (lihat Tabel 2.1).

Tabel 2.1 Waktu yang diperlukan untuk *exhaustive key search*

Ukuran kunci	Jumlah kemungkinan kunci	Lama waktu untuk 10^6 percobaan per detik	Lama waktu untuk 10^{12} percobaan per detik
16 bit	$2^{16} = 65536$	32.7 milidetik	0.0327 mikrodetik
32 bit	$2^{32} = 4.3 \times 10^9$	35.8 menit	2.15 milidetik
56 bit	$2^{56} = 7.2 \times 10^{16}$	1142 tahun	10.01 jam
128 bit	$2^{128} = 4.3 \times 10^{38}$	5.4×10^{24} tahun	5.4×10^{18} tahun

Sumber: William Stallings, *Data and Computer Communication Fourth Edition*

Untuk menghadapi serangan ini, perancang kriptosistem (kriptografer) harus membuat kunci yang panjang dan tidak mudah ditebak.

2. *Analytical attack*

   Jenis serangan ini dikenal sebagai analisis kelemahan algoritmik atau serangan analitis, yang tidak bergantung pada pencarian secara brute force untuk semua kemungkinan kunci, tetapi berfokus pada analisis struktural algoritma kriptografi yang digunakan. Dalam serangan ini, seorang kriptanalis atau penyerang tidak menjelajahi seluruh ruang kunci satu per satu, tetapi mencoba menemukan pola, properti, atau kelemahan matematis dalam algoritma itu sendiri untuk mempersempit kemungkinan ruang kunci. Tujuannya adalah untuk menghilangkan atau mengesampingkan sejumlah besar kunci yang tidak relevan, sehingga hanya

menyisakan sejumlah kecil kunci untuk diperiksa, atau bahkan untuk mengidentifikasi kunci yang benar secara langsung.

Pendekatan ini didasarkan pada pemahaman mendalam tentang algoritma kriptografi yang digunakan, termasuk pengetahuan tentang bagaimana algoritma memproses plaintext menjadi ciphertext menggunakan kunci tertentu. Kriptanalis menggunakan informasi dari definisi matematis algoritma untuk membuat persamaan atau hubungan antara komponen kriptografi, seperti ciphertext, plaintext, dan kunci. Persamaan ini dapat memberikan petunjuk penting tentang variabel tertentu yang mengungkapkan bagian dari kunci atau plaintext, tanpa harus mencoba setiap kemungkinan kunci.

Beberapa jenis analisis berbasis kelemahan kriptografi meliputi serangan diferensial, serangan linier, atau serangan aljabar. Dalam serangan diferensial, misalnya, seorang kriptanalis akan mencoba menemukan pola perbedaan dalam hasil enkripsi ketika dua teks biasa yang serupa dienkripsi dengan kunci yang sama, untuk mendapatkan informasi tentang kunci tersebut. Di sisi lain, serangan linier bergantung pada korelasi statistik yang terukur antara bit-bit tertentu dari teks biasa, teks sandi, dan kunci. Dalam serangan aljabar, seorang kriptanalis mencoba mengembangkan serangkaian persamaan yang menghubungkan komponen-komponen algoritma kriptografi, dan dengan memecahkan persamaan-persamaan ini, kunci dapat diperoleh.

Agar algoritma kriptografi dapat menghindari serangan-serangan ini, seorang kriptografer harus memastikan bahwa proses enkripsi menghasilkan hubungan

matematika yang kompleks antara teks biasa, teks sandi, dan kunci. Dengan kata lain, algoritma tersebut harus cukup "rumit" secara matematis sehingga setiap bagian dari kunci atau teks biasa tidak dapat dengan mudah diungkapkan hanya dengan memecahkan persamaan-persamaan sederhana yang berasal dari pola-pola dalam algoritma tersebut. Dalam suatu algoritma yang kuat, teks sandi harus merupakan fungsi matematika yang sangat rumit dari teks biasa dan kunci, sehingga sulit untuk memecahkan atau memodelkan algoritma dalam bentuk persamaan matematika sederhana.

Metode *analytical attack* biasanya lebih cepat menemukan kunci dibandingkan dengan *exhaustive attack*.

Data yang digunakan untuk menyerang sistem kriptografi dapat dikategorikan sebagai berikut:

1. *Chipertext only*.
2. *Known plaintext* dan *corresponding chipertext*.
3. *Chosen plaintext* dan *corresponding chipertext*.
4. *Chosen chipertext* dan *corresponding plaintext*.

Berdasarkan ketersediaan data yang ada, serangan terhadap kriptografi dapat diklasifikasikan menjadi (asumsi yang digunakan: kriptanalis mengetahui algoritma kriptografi yang digunakan):

1. *Chipertext-only attack*
   Kriptanalis memiliki beberapa cipherteks dari beberapa pesan, semuanya dienkripsi dengan algoritma yang sama. Tugas kriptanalisis adalah menemukan plainteks sebanyak mungkin atau menemukan kunci yang digunakan untuk mengenkripsi pesan.

Diberikan: $C_1 = E_k(P_1)$, $C_2 = E_k(P_2)$, …, $C_i = E_k(P_i)$ Deduksi: P_1, P_2, …, P_i atau k untuk mendapatkan P_{i+1} dari $C_{i+1} = E_k(P_{i+1})$.

2. *Known-plaintext attack*

   Beberapa pesan yang formatnya terstruktur membuka peluang kepada kriptanalis untuk menerka plainteks dari cipherteks yang bersesuaian.

   Misalnya: *From* dan *To* di dalam *e-mail*, "Dengan hormat", *wassalam*, pada surat resmi.

   #*include*, *program*, di dalam *source code*

   Diberikan: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$, …, P_i, $C_i = E_k(P_i)$

   Deduksi: k untuk mendapatkan P_{i+1} dari $C_{i+1} = E_k(P_{i+1})$.

3. *Chosen-plaintext attack*

   Serangan jenis ini lebih hebat daripada *known-plaintext attack*, karena kriptanalis dapat memilih plainteks tertentu untuk dienkripsikan, yaitu plainteks-plainteks yang lebih mengarahkan penemuan kunci.

   Diberikan: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2)$, …, P_i, $C_i = E_k(P_i)$ di mana kriptanalis dapat memilih diantara P_1, P_2, …, P_i

   Deduksi: k untuk mendapatkan P_{i+1} dari $C_{i+1} = E_k(P_{i+1})$.

4. *Adaptive-chosen-plaintext attack*

   Kasus khusus dari jenis serangan nomor 3 di atas. Misalnya, kriptanalis memilih blok plainteks yang besar, lalu dienkripsi, kemudian memilih blok lainnya yang lebih kecil berdasarkan hasil serangan sebelumnya, begitu seterusnya.

5. *Chosen-ciphertext attack*
   Kriptanalis memiliki akses terhadap cipherteks yang didekripsi (misalnya terhadap mesin elektronik yang melakukan dekripsi secara otomatis).

   Diberikan: C_1, $P_1 = D_k(C_1)$, C_2, $P_2 = D_k(P_2)$, …,
   C_i, $P_i = D_k(C_i)$
   Deduksi: k (yang mungkin diperlukan untuk mendekripsi pesan pada waktu yang akan datang) Jenis serangan ini dipakai pada algoritma kunci-publik.

6. *Chosen-text attack*
   Gabungan *chosen-plaintext attack* dan *chosen-ciphertext attack*.

7. *Chosen-key attack*
   Kriptanalis memiliki pengetahuan mengenai hubungan antara kunci-kunci yang berbeda, dan memilih kunci yang tepat untuk mendekripsi pesan.

8. *Rubber-hose cryptanalysis*
   Kriptanalis mengancam, mengirim surat gelap, atau melakukan penyiksaan sampai orang yang memegang kunci memberinya kunci untuk mendekripsi pesan. Mungkin ini cara yang terbaik untuk memecahkan kriptografi.

Kompleksitas serangan dapat diukur dengan beberapa cara:

1. Kompleksitas data (*data complexity*)
   Jumlah data yang dibutuhkan sebagai masukan untuk serangan. Semakin banyak data yang dibutuhkan untuk melakukan serangan, berarti semakin bagus algoritma kriptografi tersebut.

2. Kompleksitas waktu (*time complexity*)
   Waktu yang dibutuhkan untuk melakukan serangan. Ini disebut juga faktor kerja (*work factor*). Semakin lama waktu yang dibutuhkan untuk untuk melakukan serangan, berarti semakin bagus algoritma kriptografi tersebut.
3. Kompleksitas ruang memori (*space/storage complexity*)
   Jumlah memori yang dibutuhkan untuk melakukan serangan. Semakin banyak memori yang yang dibutuhkan untuk untuk melakukan serangan, berarti semakin bagus algoritma kriptografi tersebut.

Dalam pembahasan tentang serangan terhadap kriptografi, kita selalu mengasumsikan kriptanalis mengetahui algoritma kriptografi, sehingga keamanan algoritma terletak sepenuhnya pada kunci. Hal ini didasarkan pada *Prinsip Kerckhoff* (1883) yang berbunyi:

> Prinsip Kerckhoff: *Semua algoritma kriptografi harus publik; hanya kunci yang rahasia*.

Jika keamanan algoritma kriptografi ditentukan hanya dengan menjaga kerahasiaan algoritmanya, maka algoritma tersebut dikenal dengan nama *algoritma terbatas* (*restricted algorithm*). Algoritma terbatas mengandalkan kerahasiaan struktur dan metode operasionalnya sebagai perlindungan utama untuk menjaga data tetap aman. Namun, algoritma seperti ini sudah dianggap kurang cocok dalam sistem keamanan modern saat ini, terutama karena kelemahan inheren dalam konsep keamanannya.

Misalkan algoritma terbatas digunakan oleh orang-orang di dalam sebuah grup yang memiliki kebutuhan untuk menjaga kerahasiaan komunikasi internal mereka. Dalam skenario ini, setiap anggota grup mengenkripsi dan mendekripsi pesan dengan algoritma yang hanya diketahui oleh mereka saja. Keamanan komunikasi mereka sepenuhnya bergantung pada

fakta bahwa hanya anggota grup tersebut yang memiliki akses terhadap algoritma enkripsi. Akan tetapi, situasi menjadi rumit ketika terjadi perubahan dalam keanggotaan grup.

Sebagai contoh, jika seorang anggota grup memutuskan untuk keluar, atau dikeluarkan, maka keamanan algoritma terbatas tersebut berisiko terancam. Hal ini disebabkan oleh kenyataan bahwa mantan anggota grup tersebut kini memiliki pengetahuan tentang algoritma enkripsi yang digunakan oleh grup tersebut. Karena anggota yang keluar tidak selalu dapat dijamin untuk terus menjaga kerahasiaan algoritma grup yang mereka tinggalkan, risiko keamanan meningkat. Maka dari itu, setiap kali seorang anggota keluar dari grup, seluruh algoritma yang digunakan harus diganti atau diperbarui dengan yang baru, agar anggota yang sudah tidak lagi menjadi bagian dari grup tersebut tidak memiliki akses yang dapat membahayakan keamanan informasi di masa depan.

Dengan mempublikasikan algoritma kriptografi, kriptografer memperoleh konsultasi gratis dari sejumlah kriptologis akademisi yang ingin sekali memecahkan algoritma sehingga mereka dapat mempubliksikan paper yang memperlihatkan kecerdasan mereka.

Jika banyak pakar telah mencoba memecahkan algoritma selama 5 tahun setelah dipublikasikan dan tidak seorangpun berhasil, maka mungkin algoritma tersebut tangguh.

2.4. Keamanan Algoritma kriptografi

Sebuah algoritma kriptografi dikatakan aman (*computationally secure*) bila ia memenuhi tiga kriteria berikut:

1. Persamaan matematis yang menggambarkan operasi algoritma kriptografi sangat kompleks sehingga algoritma tidak mungkin dipecahkan secara analitik.
2. Biaya untuk memecahkan cipherteks melampaui nilai informasi yang terkandung di dalam cipherteks tersebut.

3. Waktu yang diperlukan untuk memecahkan cipherteks melampaui lamanya waktu informasi tersebut harus dijaga kerahasiaannya.

Lihat Tabel 2.1 untuk panjang kunci 128 bit (7 karakter). Untuk menemukan kunci, setidaknya setengah dari semua kemungkinan kunci yang ada harus dicoba, dan akan menghabiskan waktu 5.4×10^{24} tahun untuk satu juta percobaan per detik. Hal ini tidak mungkin karena umur alam ini saja baru pada orde 10^{11} tahun.

3.
Teori **Bilangan**

Teori bilangan (number theory) adalah teori yang mendasar dalam memahami algoritma kriptografi. Bilangan yang dimaksudkan adalah bilangan bulat (integer).

3.1. Bilangan Bulat

Bilangan bulat adalah bilangan yang tidak mempunyai pecahan desimal, misalnya 8, 21, 8765, -34, 0.

Berlawanan dengan bilangan bulat adalah bilangan riil yang mempunyai titik desimal, seperti 8.0, 34.25, 0.02.

Sifat Pembagian pada Bilangan Bulat

Misalkan a dan b adalah dua buah bilangan bulat dengan syarat $a \neq 0$. Kita menyatakan bahwa a **habis membagi** b (*a divides b*) jika terdapat bilangan bulat c sedemikian sehingga $b = ac$.

Notasi: $a \mid b$ jika $b = ac$, $c \in \mathbf{Z}$ dan $a \neq 0$. ($\mathbf{Z}$ = himpunan bilangan bulat) Kadang-kadang pernyataan "a habis membagi b" ditulis juga "b **kelipatan** a".

Contoh 1:
$4 \mid 12$ karena $12 \div 4 = 3$ (bilangan bulat) atau $12 = 4 \times 3$. Tetapi $4 \mid 13$ karena $13 \div 4 = 3.25$ (bukan bilangan bulat).

Teorema 1 (Teorema Euclidean). Misalkan m dan n adalah dua buah bilangan bulat dengan syarat $n > 0$. Jika m dibagi dengan n

maka terdapat dua buah bilangan bulat unik q (*quotient*) dan r (*remainder*), sedemikian sehingga

$$m = nq + r \quad (1) \text{ dengan } 0 \le r < n.$$

Contoh 2.

(i) 1987 dibagi dengan 97 memberikan hasil bagi 20 dan sisa 47:
$1987 = 97 \cdot 20 + 47$

(ii) –22 dibagi dengan 3 memberikan hasil bagi –8 dan sisa 2:
$-22 = 3(-8) + 2$
tetapi $-22 = 3(-7) - 1$ salah karena $r = -1$ tidak memenuhi syarat $0 \le r < n$.

3.2. Pembagi Bersama Terbesar (PBB)

Misalkan a dan b adalah dua buah bilangan bulat tidak nol. Pembagi bersama terbesar (PBB – **greatest common divisor** atau *gcd*) dari a dan b adalah bilangan bulat terbesar d sedemikian sehingga $d \mid a$ dan $d \mid b$. Dalam hal ini kita nyatakan bahwa PBB(a, b) = d.

Contoh 3. Faktor pembagi 45: 1, 3, 5, 9, 15, 45; Faktor pembagi 36: 1, 2, 3, 4, 9, 12, 18, 36; Faktor pembagi bersama dari 45 dan 36 adalah 1, 3, 9 PBB(45, 36) = 9.

Algoritma Euclidean

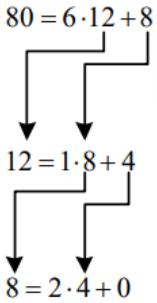
Algoritma Euclidean adalah algoritma untuk mencari PBB dari dua buah bilangan bulat. Euclid, penemu algoritma Euclidean, adalah seorang matematikawan Yunani yang menuliskan algoritmanya tersebut dalam bukunya yang terkenal, *Element*.

Diberikan dua buah bilangan bulat tak-negatif m dan n ($m \ge n$). Algoritma Euclidean berikut mencari pembagi bersama terbesar dari m dan n.

Algoritma Euclidean

1. Jika $n = 0$ maka m adalah PBB(m, n); stop.
   tetapi jika $n \neq 0$, lanjutkan ke langkah 2.
2. Bagilah m dengan n dan misalkan r adalah sisanya.
3. Ganti nilai m dengan nilai n dan nilai n dengan nilai r, lalu ulang kembali ke langkah 1.

Contoh 4. $m = 80$, $n = 12$ dan dipenuhi syarat $m \geq n$



Sisa pembagian terakhir sebelum 0 adalah 4, maka PBB(80, 12)= 4

3.3. Relatif Prima

Dua buah bilangan bulat a dan b dikatakan *relatif prima* jika PBB(a, b) = 1.

Contoh 5.

20 dan 3 relatif prima sebab PBB(20, 3) = 1. Begitu juga 7 dan 11 relatif prima karena PBB(7, 11) = 1. Tetapi 20 dan 5 tidak relatif prima sebab PBB(20, 5) = 5 ≠ 1.

Jika a dan b relatif prima, maka terdapat bilangan bulat m dan n sedemikian sehingga

$$ma + nb = 1$$

Contoh 6.

Bilangan 20 dan 3 adalah relatif prima karena PBB(20, 3) =1, atau dapat ditulis

$$2 \cdot 20 + (-13) \cdot 3 = 1$$

dengan $m = 2$ dan $n = -13$. Tetapi 20 dan 5 tidak relatif prima karena PBB(20, 5) = 5 ≠ 1 sehingga 20 dan 5 tidak dapat dinyatakan dalam $m \cdot 20 + n \cdot 5 = 1$.

3.4. Aritmetika Modulo

Misalkan a adalah bilangan bulat dan m adalah bilangan bulat > 0. Operasi a mod m (dibaca "a modulo m") memberikan sisa jika a dibagi dengan m. Notasi: $a \bmod m = r$ sedemikian sehingga $a = mq + r$, dengan $0 \le r < m$.

Bilangan m disebut **modulus** atau **modulo**, dan hasil aritmetika modulo m terletak di dalam himpunan {0, 1, 2, …, m – 1} (mengapa?).

Contoh 7.
Beberapa hasil operasi dengan operator modulo:

(i) 23 mod 5 = 3 (23 = 5 · 9 + 0)
(ii) 27 mod 3 = 0 (27 = 3 · 9 + 0)
(iii) 6 mod 8= 6 (6 = 8 · 0 + 6)
(iv) 0 mod 12 = 0 (0 = 12 · 0 + 0)
(v) -41 mod 9 = 4 (-41 = 9(-5) + 4)
(vi) -39 mod 13 = 0 (-39 = 13(-3) + 0)

Penjelasan (v): Karena a negatif, bagi $|a|$ dengan m mendapatkan sisa r'. Maka $a \bmod m = m - r'$ bila $r' \neq 0$. Jadi $|-41| \bmod 9 = 5$, sehingga $-41 \bmod 9 = 9 - 5 = 4$.

Kongruen

- Misalnya 38 mod 5 = 3 dan 13 mod 5 = 3, maka kita katakan 38 ≡ 13 (mod 5) (baca: 38 kongruen dengan 13 dalam modulo 5).
- Misalkan a dan b adalah bilangan bulat dan m adalah bilangan > 0, maka $a \equiv b \pmod{m}$ jika m habis membagi $a - b$.
- Jika a tidak kongruen dengan b dalam modulus m, maka ditulis $a \not\equiv b \pmod{m}$.

Contoh 8.

$17 \equiv 2 \pmod{3}$	(3 habis membagi 17 – 2 = 15)
$-7 \equiv 15 \pmod{11}$	(11 habis membagi –7 – 15 = –22)
$12 \not\equiv 2 \pmod{7}$	(7 tidak habis membagi 12 – 2 = 10)
$-7 \not\equiv 15 \pmod{3}$	(3 tidak habis membagi –7 – 15 = –22)

Kekongruenan $a \equiv b \pmod{m}$ dapat pula dituliskan dalam hubungan
$a = b + km$ yang dalam hal ini k adalah bilangan bulat.

Contoh 9.
$17 \equiv 2 \pmod{3}$ dapat ditulis sebagai $17 = 2 + 5 \cdot 3$
$-7 = 15 \pmod{11}$ dapat ditulis sebagai $-7 = 15 + (-2)11$

Berdasarkan definisi aritmetika modulo, kita dapat menuliskan $a \bmod m = r$ sebagai $a \equiv r \pmod{m}$

Contoh 10.
Beberapa hasil operasi dengan operator modulo berikut:

(i) $23 \bmod 5 = 3$ dapat ditulis sebagai $23 \equiv 3 \pmod{5}$
(ii) $27 \bmod 3 = 0$ dapat ditulis sebagai $27 \equiv 0 \pmod{3}$
(iii) $6 \bmod 8 = 6$ dapat ditulis sebagai $6 \equiv 6 \pmod{8}$
(iv) $0 \bmod 12 = 0$ dapat ditulis sebagai $0 \equiv 0 \pmod{12}$

(v) - 41 mod 9 = 4 dapat ditulis sebagai -41 ≡ 4 (mod 9)
(vi) - 39 mod 13 = 0 dapat ditulis sebagai -39 ≡ 0 (mod 13)

Teorema 2. Misalkan m adalah bilangan bulat positif.

1. Jika $a \equiv b \pmod m$ dan c adalah sembarang bilangan bulat maka
   (i) $(a + c) \equiv (b + c) \pmod m$
   (ii) $ac \equiv bc \pmod m$
   (iii) $a^p \equiv b^p \pmod m$ untuk suatu bilangan bulat tak negatif p.

2. Jika $a \equiv b \pmod m$ dan $c \equiv d \pmod m$, maka
   (i) $(a + c) \equiv (b + d) \pmod m$
   (ii) $ac \equiv bd \pmod m$

Bukti (hanya untuk 1(ii) dan 2(i) saja):

1(ii) $a \equiv b \pmod m$ berarti:

$\Leftrightarrow a = b + km$
$\Leftrightarrow a - b = km$
$\Leftrightarrow (a - b)c = ckm$
$\Leftrightarrow ac = bc + Km$
$\Leftrightarrow ac \equiv bc \pmod m$

2(i) $a \equiv b \pmod m \quad \Leftrightarrow a = b + k_1 m$
$c \equiv d \pmod m \quad \Leftrightarrow c = d + k_2 m$ +
$\Leftrightarrow (a + c) = (b + d) + (k_1 + k_2)m$
$\Leftrightarrow (a + c) = (b + d) + km \quad (k = k_1 + k_2)$
$\Leftrightarrow (a + c) = (b + d) \pmod m$

Contoh 11.

Misalkan 17 ≡ 2 (mod 3) dan 10 ≡ 4 (mod 3), maka menurut Teorema 2,

17 + 5 = 2 + 5 (mod 3) ⇔ 22 = 7 (mod 3)

17. 5 = 5 · 2 (mod 3) ⇔ 85 = 10 (mod 3)
17 + 10 = 2 + 4 (mod 3) ⇔ 27 = 6 (mod 3)
17. 10 = 2 · 4 (mod 3) ⇔ 170 = 8 (mod 3)

Perhatikanlah bahwa Teorema 2 tidak memasukkan operasi pembagian pada aritmetika modulo karena jika kedua ruas dibagi dengan bilangan bulat, maka kekongruenan tidak selalu dipenuhi. Misalnya:

(i) $10 \equiv 4 \pmod 3$ dapat dibagi dengan 2 karena $10/2 = 5$ dan $4/2 = 2$, dan $5 \equiv 2 \pmod 3$
(ii) $14 \equiv 8 \pmod 6$ tidak dapat dibagi dengan 2, karena $14/2 = 7$ dan $8/2 = 4$, tetapi $7 \not\equiv 4 \pmod 6$.

Balikan Modulo (modulo invers)
Jika a dan m relatif prima dan $m > 1$, maka kita dapat menemukan balikan (*invers*) dari a modulo m. Balikan dari a modulo m adalah bilangan bulat a sedemikian sehingga
$aa\_ \equiv 1 \pmod m$

Bukti: Dari definisi relatif prima diketahui bahwa PBB(a, m) = 1, dan menurut persamaan (2) terdapat bilangan bulat p dan q sedemikian sehingga
$pa + qm = 1$

yang mengimplikasikan bahwa
$pa + qm \equiv 1 \pmod m$

Karena $qm \equiv 0 \pmod m$, maka
$pa \equiv 1 \pmod m$

Kekongruenan yang terakhir ini berarti bahwa p adalah balikan dari a modulo m.

Pembuktian di atas juga menceritakan bahwa untuk mencari balikan dari a modulo m, kita harus membuat kombinasi lanjar dari a dan m sama dengan 1. Koefisien a dari kombinasi lanjar tersebut merupakan balikan dari a modulo m.

Contoh 12.
Tentukan balikan dari 4 (mod 9), 17 (mod 7), dan 18 (mod 10).
Penyelesaian:

a) Karena PBB(4, 9) = 1, maka balikan dari 4 (mod 9) ada. Dari algoritma Euclidean diperoleh bahwa
$9 = 2 \cdot 4 + 1$

Susun persamaan di atas menjadi
$-2 \cdot 4 + 1 \cdot 9 = 1$

Dari persamaan terakhir ini kita peroleh –2 adalah balikan dari 4 modulo 9. Periksalah bahwa
$-2 \cdot 4 \equiv 1 \pmod 9$ (9 habis membagi $-2 \cdot 4 - 1 = -9$)

b) Karena PBB(17, 7) = 1, maka balikan dari 17 (mod 7) ada. Dari algoritma Euclidean diperoleh rangkaian pembagian berikut
$17 = 2 \cdot 7 + 3$ (i)
$7 = 2 \cdot 3 + 1$ (ii)
$3 = 3 \cdot 1 + 0$ (iii) (yang berarti: PBB(17, 7) = 1))
Susun (ii) menjadi:
$1 = 7 - 2 \cdot 3$ (iv)

Susun (i) menjadi
$3 = 17 - 2 \cdot 7$ (v)

Sulihkan (v) ke dalam (iv):

$1 = 7 - 2 \cdot (17 - 2 \cdot 7) = 1 \cdot 7 - 2 \cdot 17 + 4 \cdot 7 = 5 \cdot 7 - 2 \cdot 17$

atau

$-2 \cdot 17 + 5 \cdot 7 = 1$

Dari persamaan terakhir ini kita peroleh –2 adalah balikan dari 17 modulo 7.

$-2 \cdot 17 \equiv 1 \pmod 7$ (7 habis membagi $-2 \cdot 17 - 1 = -35$)

c) Karena PBB(18, 10) = $2 \neq 1$, maka balikan dari 18 (mod 10) tidak ada.

Kekongruenan Lanjar

Kekongruenan lanjar adalah kongruen yang berbentuk

$$ax \equiv b \pmod m$$

dengan m adalah bilangan bulat positif, a dan b sembarang bilangan bulat, dan x adalah peubah bilangan bulat.

Nilai-nilai x dicari sebagai berikut:

$$ax = b + km$$

Yang Dapat Di susun menjadi

$$x = \frac{b + km}{a}$$

dengan k adalah sembarang bilangan bulat. Cobakan untuk k = 0, 1, 2, … dan k = –1, –2, … yang menghasilkan x sebagai bilangan bulat.

Contoh 13.

Tentukan solusi: $4x \equiv 3 \pmod 9$ dan $2x \equiv 3 \pmod 4$

Penyelesaian:

(i) $4x \equiv 3 \pmod 9$

$$x = \frac{3 + k \cdot 9}{4}$$

$k = 0 \rightarrow x = (3 + 0 \cdot 9)/4 = 3/4$ (bukan solusi)
$k = 1 \rightarrow x = (3 + 1 \cdot 9)/4 = 3$
$k = 2 \rightarrow x = (3 + 2 \cdot 9)/4 = 21/4$ (bukan solusi)
$k = 3,\ k = 4$ tidak menghasilkan solusi
$k = 5 \rightarrow x = (3 + 5 \cdot 9)/4 = 12$
...
$k = -1 \rightarrow x = (3 - 1 \cdot 9)/4 = -6/4$ (bukan solusi)
$k = -2 \rightarrow x = (3 - 2 \cdot 9)/4 = -15/4$ (bukan solusi)
$k = -3 \rightarrow x = (3 - 3 \cdot 9)/4 = -6$
...
$k = -6 \rightarrow x = (3 - 6 \cdot 9)/4 = -15$
...

Nilai-nilai x yang memenuhi: 3, 12, ... dan –6, –15, ...

(ii) $2x \equiv 3 \pmod 4$

$$x = \frac{3 + k \cdot 4}{2}$$

Karena $4k$ genap dan 3 ganjil maka penjumlahannya menghasilkan ganjil, sehingga hasil penjumlahan tersebut jika dibagi dengan 2 tidak menghasilkan bilangan bulat. Dengan kata lain, tidak ada nilai-nilai x yang memenuhi $2x \equiv 3 \pmod 4$.

Chinese Remainder Problem

Pada abad pertama, seorang matematikawan China yang bernama Sun Tse mengajukan pertanyaan sebagai berikut:

Tentukan sebuah bilangan bulat yang bila dibagi dengan 5 menyisakan 3, bila dibagi 7 menyisakan 5, dan bila dibagi 11 menyisakan 7.

Pertanyaan Sun Tse dapat dirumuskan kedalam sistem kongruen lanjar:

$x \equiv 3 \pmod 5$

$x \equiv 5 \pmod 7$

$x \equiv 7 \pmod{11}$

TEOREMA 5.6. (*Chinese Remainder Theorem*) Misalkan m_1, m_2, …, m_n adalah bilangan bulat positif sedemikian sehingga PBB(m_i, m_j) = 1 untuk $i \neq j$. Maka sistem kongruen lanjar

$$x \equiv a_k \pmod{m_k}$$

mempunyai sebuah solusi unik modulo $m = m_1 \cdot m_2 \cdot \ldots \cdot m_n$.

Contoh 14.

Tentukan solusi dari pertanyaan Sun Tse di atas.

Penyelesaian:

Menurut persamaan (5.6), kongruen pertama, $x \equiv 3 \pmod 5$, memberikan $x = 3 + 5k_1$ untuk beberapa nilai k. Sulihkan ini ke dalam kongruen kedua menjadi $3 + 5k_1 \equiv 5 \pmod 7$, dari sini kita peroleh $k_1 \equiv 6 \pmod 7$, atau $k_1 = 6 + 7k_2$ untuk beberapa nilai k_2. Jadi kita mendapatkan $x = 3 + 5k_1 = 3 + 5(6 + 7k_2) = 33 + 35k_2$ yang mana memenuhi dua kongruen pertama. Jika x memenuhi kongruen yang ketiga, kita harus mempunyai $33 + 35k_2 \equiv 7 \pmod{11}$, yang mengakibatkan $k_2 \equiv 9 \pmod{11}$ atau $k_2 = 9 + 11k_3$. Sulihkan k_2 ini ke dalam kongruen yang ketiga menghasilkan $x = 33 + 35(9 + 11k_3) \equiv 348 + 385k_3 \pmod{11}$. Dengan demikian, $x \equiv 348 \pmod{385}$ yang memenuhi ketiga konruen tersebut. Dengan kata lain, 348 adalah solusi unik modulo 385. Catatlah bahwa $385 = 5 \cdot 7 \cdot 11$.

Solusi unik ini mudah dibuktikan sebagai berikut. Solusi tersebut modulo $m = m_1 \cdot m_2 \cdot m_3 = 5 \cdot 7 \cdot 11 = 5 \cdot 77 = 11 \cdot 35$. Karena $77\ \ 3 \equiv 1 \pmod 5$, $55 \cdot 6 \equiv 1 \pmod 7$, dan $35 \cdot 6 \equiv 1 \pmod{11}$, solusi unik dari sistem kongruen tersebut adalah

$$\begin{aligned} x \ &\equiv 3 \cdot 77 \cdot 3 + 5 \cdot 55 \cdot 6 \ + 7 \cdot 35 \cdot 6 \pmod{385} \\ &\equiv 3813 \pmod{385} \equiv 348 \pmod{385} \end{aligned}$$

3.5. Aritmetika Modulo dan Kriptografi

Aritmetika modulo cocok digunakan untuk kriptografi karena dua alasan:

1. Oleh karena nilai-nilai aritmetika modulo berada dalam himpunan berhingga (0 sampai modulus m – 1), maka kita tidak perlu khawatir hasil perhitungan berada di luar himpunan.
2. Karena kita bekerja dengan bilangan bulat, maka kita tidak khawatir kehilangan informasi akibat pembulatan (*round off*) sebagaimana pada operasi bilangan riil.

3.6. Bilangan Prima

- Bilangan bulat positif p ($p > 1$) disebut bilangan prima jika pembaginya hanya 1 dan p.
- Contoh: 23 adalah bilangan prima karena ia hanya habis dibagi oleh 1 dan 23.
- Karena bilangan prima harus lebih besar dari 1, maka barisan bilangan prima dimulai dari 2, yaitu 2, 3, 5, 7, 11, 13, …. Seluruh bilangan prima adalah bilangan ganjil, kecuali 2 yang merupakan bilangan genap.
- Bilangan selain prima disebut bilangan **komposit** (*composite*). Misalnya 20 adalah bilangan komposit karena 20 dapat dibagi oleh 2, 4, 5, dan 10, selain 1 dan 20 sendiri.

Teorema 3. (*The Fundamental Theorem of Arithmetic*).
Setiap bilangan bulat positif yang lebih besar atau sama dengan 2 dapat dinyatakan sebagai perkalian atau lebih bilangan bulay

Contoh 15.

9 = 3 x 3		(2 buah faktor prima)
100 = 2 x 2 x 5 x 5		(4 buah faktor prima)
13 = 13	(atau 1 x 13)	(1 buah faktor prima)

- Untuk menguji apakah n merupakan bilangan prima atau komposit, kita cukup membagi n dengan sejumlah bilangan prima, mulai dari 2, 3, … , bilangan prima ≤ $\sqrt{n}$. Jika n habis dibagi dengan salah satu dari bilangan prima tersebut, maka n adalah bilangan komposit, tetapi jika n tidak habis dibagi oleh semua bilangan prima tersebut, maka n adalah bilangan prima.

Contoh 16.
Tunjukkan apakah (i) 171 dan (ii) 199 merupakan bilangan prima atau komposit.
Penyelesaian:

(i) $\sqrt{171}$ = 13.077. Bilangan prima yang ≤ $\sqrt{171}$ adalah 2, 3, 5, 7, 11, 13. Karena 171 habis dibagi 3, maka 171 adalah bilangan komposit.

(ii) $\sqrt{199}$ = 14.107. Bilangan prima yang ≤ $\sqrt{199}$ adalah 2, 3, 5, 7, 11, 13. Karena 199 tidak habis dibagi 2, 3, 5, 7, 11, dan 13, maka 199 adalah bilangan prima.

- Terdapat metode lain yang dapat digunakan untuk menguji keprimaan suatu bilangan bulat, yang terkenal dengan **Teorema Fermat**. Fermat (dibaca "Fair-ma") adalah seorang matematikawan Perancis pada tahun 1640.

Teorema 4 (**Teorema Fermat**). Jika p adalah bilangan prima dan a adalah bilangan bulat yang tidak habis dibagi dengan p, yaitu PBB(a, p) = 1, maka

$$a^{p-1} \equiv 1 \pmod{p}$$

Contoh 17.
Kita akan menguji apakah 17 dan 21 bilangan prima atau bukan. Di sini kita mengambil nilai a = 2 karena PBB(17, 2) = 1 dan PBB(21, 2) = 1. Untuk 17,
$2^{17-1} = 65536 \equiv 1 \pmod{17}$

karena 17 tidak membagi 65536 – 1 = 65535 (65535÷17 = 3855).
Untuk 21,
$2^{21-1} = 1048576 \equiv \backslash\ 1 \pmod{21}$

karena 21 tidak habis membagi 1048576 – 1 = 1048575.

- Kelemahan Teorema Fermat: terdapat bilangan komposit n sedemikian sehingga $2^{n-1} \equiv 1 \pmod{n}$. Bilangan bulat seperti itu disebut bilangan **prima semu** (*pseudoprimes*).
- Misalnya komposit 341 (yaitu 341 = 11 · 31) adalah bilangan prima semu karena menurut teorema Fermat,

$$2^{340} \equiv 1 \pmod{341}$$

Untunglah bilangan prima semu relatif jarang terdapat.

Fungsi Euler f
Fungsi Euler φ medefinisikan φ(n) untuk $n \geq 1$ yang menyatakan jumlah bilangan bulat positif < n yang relatif prima dengan n.

Contoh 18.
Tentukan φ(20).
Penyelesaian:

Bilangan bulat positif yang lebih kecil dari 20 adalah 1 sampai 19. Di antara bilangan-bilangan tersebut, terdapat φ(20) = 8 buah yang relatif prima dengan 20, yaitu 1, 3, 7, 9, 11, 13, 17, 19.

Untuk n = 1, 2, …, 10, fungsi Euler adalah

$\phi(1) = 0$	$\phi(6) = 2$
$\phi(2) = 1$	$\phi(7) = 6$
$\phi(3) = 2$	$\phi(8) = 4$
$\phi(4) = 2$	$\phi(9) = 6$
$\phi(5) = 4$	$\phi(10) = 4$

Jika n prima, maka setiap bilangan bulat yang lebih kecil dari n relatif prima terhadap n. Dengan kata lain, $\varphi(n) = n - 1$ hanya jika n prima.

Contoh 19

φ(3) = 2, φ(5) = 4, φ(7) = 6, φ(11) = 10, φ(13) = 12, dst.

Teorema 5. Jika $n = pq$ adalah bilangan komposit dengan p dan q prima, maka $\varphi(n) = \varphi(p)\,\varphi(q) = (p-1)(q-1)$.

Contoh 20.

Tentukan φ(21).

Penyelesaian:

Karena $21 = 7 \cdot 3$, $\varphi(21) = \varphi(7)\,\varphi(3) = 6 \cdot 2 = 12$ buah bilangan bulat yang relatif prima terhadap 21, yaitu 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20.

Teorema 6. Jika p bilangan prima dan $k > 0$, maka $\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$.

Contoh 22.

Tentukan φ(16).

Penyelesaian:

Karena $\varphi(16) = \varphi(2^4) = 2^4 - 2^3 = 16 - 8 = 8$, maka ada delapan buah bilangan bulat yang relatif prima terhadap 16, yaitu 1, 3, 5, 7, 9, 11, 13, 15.

Teorema 7 (*Euler's generalization of Fermat theorem*). Jika PBB(a, n) = 1, maka $a^{\varphi(n)} \bmod n = 1$ (atau $a^{\varphi(n)} \equiv 1 \pmod{n}$)

4.
Algoritma **Kriptografi Klasik**

4.1. Pendahuluan

Sebelum komputer ada, kriptografi dilakukan dengan algoritma berbasis karakter. Algoritma yang digunakan termasuk ke dalam sistem kriptografi simetri dan digunakan jauh sebelum sistem kriptografi kunci publik ditemukan.

Terdapat sejumlah algoritma yang tercatat dalam sejarah kriptografi (sehingga dinamakan algoritma kriptografi klasik), namun sekarang algoritma tersebut sudah usang karena ia sangat mudah dipecahkan.

Tiga alasan mempelajari algoritma kriptografi klasik:

1. Untuk memberikan pemahaman konsep dasar kriptografi.
2. Dasar dari algoritma kriptografi modern.
3. Dapat memahami potensi-potensi kelemahan sistem *cipher*.

Algoritma kriptografi klasik:

1. *Cipher* Substitusi (*Substitution Ciphers*)
2. *Cipher* Transposisi (*Transposition Ciphers*)

Keterangan: *cipher* = algoritma kriptografi

4.2. Cipher Substitusi

Ini adalah algoritma kriptografi yang mula-mula digunakan oleh kaisar Romawi, Julius Caesar (sehingga dinamakan juga

caesar cipher), untuk menyandikan pesan yang ia kirim kepada para gubernurnya. Caranya adalah dengan mengganti (menyulih atau mensubstitusi) setiap karakter dengan karakter lain dalam susunan abjad (alfabet).

Misalnya, tiap huruf disubstitusi dengan huruf ketiga berikutnya dari susunan akjad. Dalam hal ini kuncinya adalah jumlah pergeseran huruf (yaitu $k = 3$).

Tabel substitusi:

p_i :	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
c_i :	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Contoh 1. Pesan

AWASI ASTERIX DAN TEMANNYA OBELIX

Di samarkan (enskripsi) menjadi

DZDVL DVWHULA GDQ WHPDQQBA REHOLA

Penerima pesan men-dekripsi cipherteks dengan menggunakan tabel substitusi, sehingga cipherteks

DZDVL DVWHULA GDQ WHPDQQBA REHOLA

dapat dikembalikan menjadi plainteks semula:

AWASI ASTERIX DAN TEMANNYA OBELIX

Dengan mengkodekan setiap huruf abjad dengan *integer* sebagai berikut: $A = 0$, $B = 1$, …, $Z = 25$, maka secara matematis *caesar cipher* menyandikan plainteks p_i menjadi c_i dengan aturan:

$$c_i = E(p_i) = (p_i + 3) \bmod 26 \quad (1)$$

dan dekripsi cipherteks c_i menjadi p_i dengan aturan:

$$p_i = D(c_i) = (c_i - 3) \bmod 26 \quad (2)$$

Karena hanya ada 26 huruf abjad, maka pergeseran huruf yang mungkin dilakukan adalah dari 0 sampai 25. Secara umum, untuk pergeseran huruf sejauh k (dalam hal ini k adalah kunci enkripsi dan deksripsi), fungsi enkripsi adalah

$$c_i = E(p_i) = (p_i + k) \bmod 26 \quad (3)$$

dan fungsi dekripsi adalah

$$p_i = D(c_i) = (c_i - k) \bmod 26 \quad (4)$$

Catatan:

1. Pergeseran 0 sama dengan pergeseran 26 (susunan huruf tidak berubah)
2. Pergeseran lain untuk k > 25 dapat juga dilakukan namun hasilnya akan kongruen dengan bilangan bulat dalam modulo 26. Misalnya k = 37 kongruen dengan 11 dalam modulo 26, atau 37 ≡ 11 (mod 26).
3. Karena ada operasi penjumlahan dalam persamaan (3) dan (4), maka *caesar cipher* kadang-kadang dinamakan juga *additive cipher*.

Untuk mengenkripsi/dekripsi pesan yang disusun oleh karakter-karakter teks (ASCII, 256 karakter), maka persamaan 3 dan 4 ditulis

$$c_i = E(p_i) = (p_i + k) \bmod 256 \quad (5)$$

$$p_i = D(c_i) = (c_i - k) \bmod 256 \quad (6)$$

Program Pascal sederhana untuk mengenkripsi dan dekripsi pesan teks (*text file*) dengan *caesar cipher*:

```
program enkripsi;
{ Mengenkripsi berkas 'plain.txt'
  menjadi 'cipher.txt' dengan
  metode caesar cipher }
uses
  crt;
var
   F1, F2 : text;
   p : char;
   c : integer;
   k : integer;

begin
   assign(F1, 'plain.txt');
   reset(F1);

   assign(F2, 'cipher.txt');
   rewrite(F2);

   write('k = ?'); readln(k);
   while not EOF(F1) do
    begin
      while not EOLN(F1) do
       begin
         read(F1, p);
         c := (ord(p) + k) mod 256;
         write(F2, chr(c));
       end;
      readln(F1);
      writeln(F2);
   end;
   close(F1);
   close(F2);
end.
```

```
program dekripsi;
{ Mendekripsi berkas 'cipher.txt'
  menjadi 'plain2.txt' dengan
  metode caesar cipher }
uses
  crt;
var
   F1, F2 : text;
   p : char;
   c : integer;
   k : integer;

begin
   assign(F1, 'cipher.txt');
   reset(F1);

   assign(F2, 'plain2.txt');
   rewrite(F2);

   write('k = ?'); readln(k);
   while not EOF(F1) do
    begin
      while not EOLN(F1) do
       begin
         read(F1, p);
         c := (ord(p) - k) mod 256;
         write(F2, chr(c));
       end;
      readln(F1);
      writeln(F2);
   end;
   close(F1);
   close(F2)
end.
```

Kriptanalisis Terhadap Caesar Cipher

Caesar cipher mudah dipecahkan dengan metode *exhaustive key search* karena jumlah kuncinya sangat sedikit (hanya ada 26 kunci).

Contoh 2.

Misalkan kriptanalis menemukan potongan cipherteks (disebut juga *cryptogram*) XMZVH. Diandaikan kriptanalis mengetahui bahwa plainteks disusun dalam Bahasa Inggris dan algoritma kriptografi yang digunakan adalah *caesar cipher*. Untuk memperoleh plainteks, lakukan dekripsi mulai dari kunci yang terbesar, 25, sampai kunci yang terkecil, 1. Periksa apakah

dekripsi menghasilkan pesan yang mempunyai makna (lihat Tabel 1).

Tabel 4.1 Contoh *exhaustive key search* terhadap cipherteks XMZVH

Kunci (k) *ciphering*	'Pesan' hasil dekripsi	Kunci (k) *ciphering*	'Pesan' hasil dekripsi	Kunci (k) *ciphering*	'Pesan' hasil dekripsi
0	XMZVH	17	GVIEQ	8	PERNZ
25	YNAWI	16	HWJFR	7	QFSOA
24	ZOBXJ	15	IXKGS	6	RGTPB
23	APCYK	14	JYLHT	5	SHUQC
22	BQDZL	13	KZMIU	4	TIVRD
21	CREAM	12	LANJV	3	UJWSE
20	DSFBN	11	MBOKW	2	VKXTF
19	ETGCO	10	NCPLX	1	WLYUG
18	FUHDP	9	ODQMY		

Dari Tabel 4.1, kata dalam Bahasa Inggris yang potensial menjadi plainteks adalah CREAM dengan menggunakan k = 21. Kunci ini digunakan untuk mendekripsikan cipherteks lainnya.

Kadang-kadang satu kunci yang potensial menghasilkan pesan yang bermakna tidak selalu satu buah. Untuk itu, kita membutuhkan informasi lainnya, misalnya konteks pesan tersebut atau mencoba mendekripsi potongan cipherteks lain untuk memperoleh kunci yang benar.

Contoh 3.
Misalkan potongan cipherteks **HSPPW** menghasilkan dua kemungkinan kunci yang potensial, yaitu k = 4 menghasilkan pesan DOLLS dan k = 11 menghasilkan WHEEL. Lakukan deksripsi terhadap potongan cipherteks lain tetapi hanya menggunakan k = 4 dan k = 11 (tidak perlu *exhaustive key search*) agar dapat disimpulkan kunci yang benar.

Cara lain yang digunakan untuk memecahkan cipherteks adalah dengan statistik, yaitu dengan menggunakan tabel kemunculan karakter, yang membantu mengidentifikasi karakter plainteks

yang berkoresponden dengan karakter di dalam cipherteks (akan dijelaskan kemudian).

Jenis-jenis Cipher Subsbtitusi

a. Cipher abjad-tunggal (monoalphabetic cipher atau cipher substitusi sederhana - simple substitution cipher)

Satu karakter di plainteks diganti dengan satu karakter yang bersesuaian. Jadi, fungsi *ciphering*-nya adalah fungsi satu-kesatu.

Jika plainteks terdiri dari huruf-huruf abjad, maka jumlah kemungkinan susunan huruf-huruf cipherteks yang dapat dibuat adalah sebanyak

26! = 403.291.461.126.605.635.584.000.000

Caesar cipher adalah kasus khusus dari *cipher* abjad tunggal di mana susunan huruf cipherteks diperoleh dengan menggeser huruf-huruf alfabet sejauh 3 karakter.

ROT13 adalah program enkripsi sederhana yang terdapat di dalam sistem UNIX. ROT13 menggunakan *cipher* abjad-tunggal dengan pergeseran k = 13 (jadi, huruf A diganti dengan **N**, B diganti dengan **O**, dan seterusnya).

Enkripsi arsip dua kali dengan ROT13 menghasilkan arsip semula:

P = ROT13 (ROT13 (P))

b. Cipher substitusi homofonik (Homophonic substitution cipher)

Seperti *cipher* abjad-tunggal, kecuali bahwa setiap karakter di dalam plainteks dapat dipetakan ke dalam salah satu dari karakter cipherteks yang mungkin. Misalnya huruf A dapat berkoresponden dengan **7**, **9**, atau **16**, huruf B dapat berkoresponden dengan **5**, **10**, atau **23** dan seterusnya.

Fungsi *ciphering*-nya memetakan satu-ke-banyak (*one-tomany*). Cipher substitusi homofonik digunakan pertama kali pada tahun 1401 oleh wanita bangsawan Mantua.

Cipher substitusi homofonik lebih sulit dipecahkan daripada *cipher* abjad-tunggal. Namun, dengan *known-plaintext attack*, cipher ini dapat dipecahkan, sedangkan dengan *ciphertext-only attack* lebih sulit.

c. *Cipher* abjad-majemuk (*Polyalpabetic substitution cipher*)

Merupakan *cipher* substitusi-ganda (*multiple-substitution cipher*) yang melibatkan penggunaan kunci berbeda.

Cipher abjad-majemuk dibuat dari sejumlah *cipher* abjadtunggal, masing-masing dengan kunci yang berbeda. Kebanyakan *cipher* abjad-majemuk adalah *cipher* substitusi periodik yang didasarkan pada periode *m*.

Misalkan plainteks *P* adalah

$$P = p_1 p_2 \ldots p_m p_{m+1} \ldots p_{2m} \ldots$$

maka cipherteks hasil enkripsi adalah

$$E_k(P) = f_1(p_1)\ f_2(p_2) \ldots f_m(p_m)\ f_{m+1}(p_{m+1}) \ldots f_{2m}(p_{2m}) \ldots$$

yang dalam hal ini p_i adalah huruf-huruf di dalam plainteks.

Untuk $m = 1$, *cipher*-nya ekivalen dengan *cipher* abjadtunggal.

Contoh *cipher* substitusi periodik adalah cipher Vigenere yang ditemukan oleh kriptologi Perancis, Blaise de Vigenere pada abad 16. Misalkan *K* adalah deretan kunci

$$K = k_1\ k_2 \ldots k_m$$

yang dalam hal ini k_i untuk $1 \leq i \leq m$ menyatakan jumlah pergeseran pada huruf ke-i. Maka, karakter cipherteks $y_i(p)$ adalah

$$y_i(p) = (p + k_i) \bmod n \qquad (5)$$

Misalkan periode m = 20, maka 20 karakter pertama dienkripsi dengan persamaan (5), dimana setiap karakter ke-i menggunakan kunci k_i. Untuk 20 karakter berikutnya, kembali menggunakan pola enkripsi yang sama.

Cipher abjad-majemuk ditemukan pertama kali oleh Leon Battista pada tahun 1568. Metode ini digunakan oleh tentara AS selama Perang Sipil Amerika.

Meskipun *cipher* abjad-majemuk dapat dipecahkan dengan mudah (dengan bantuan komputer), namun anehnya banyak program keamanan komputer (*computer security*) yang menggunakan *cipher* jenis ini.

d. Cipher substitusi poligram (Polygram substitution cipher)
Blok karakter disubstitusi dengan blok cipherteks. Misalnya ABA diganti dengan **RTQ**, ABB diganti dengan **SLL**, dan lain-lain.

Playfair cipher, ditemukan pada tahun 1854, termasuk ke dalam *cipher* substitusi poligram dan digunakan oleh negara Inggris selama Perang Dunia I.

4.3. *Cipher* Transposisi

Pada *cipher* transposisi, plainteks tetap sama, tetapi urutannya diubah. Dengan kata lain, algoritma ini melakukan *transpose* terhadap rangkaian karakter di dalam teks.

Nama lain untuk metode ini adalah **permutasi**, karena *transpose* setiap karakter di dalam teks sama dengan mempermutasikan karakter-karakter tersebut.

Contoh 4. Misalkan plainteks adalah

```
DEPARTEMEN TEKNIK INFORMATIKA ITB
```

Untuk meng-enkripsi pesan, plainteks ditulis secara horizontal dengan lebar kolom tetap, misal selebar 6 karakter (kunci k = 6):

```
DEPART
EMENTE
KNIKIN
FORMAT
IKAITB
```

maka cipherteksnya dibaca secara vertikal menjadi

```
DEKFIEMNOKPEIRAANKMIRTIATTENTB
```

Untuk mendekripsi pesan, kita membagi panjang cipherteks dengan kunci. Pada contoh ini, kita membagi 30 dengan 6 untuk mendapatkan 5.
Algoritma dekripsi identik dengan algoritma enkripsi. Jadi, untuk contoh ini, kita menulis cipherteks dalam baris-baris selebar 5 karakter menjadi:

```
DEKFI
EMNOK
PEIRA
ANKMI
RTIAT
TENTB
```

Dengan membaca setiap kolom kita memperoleh pesan semula:

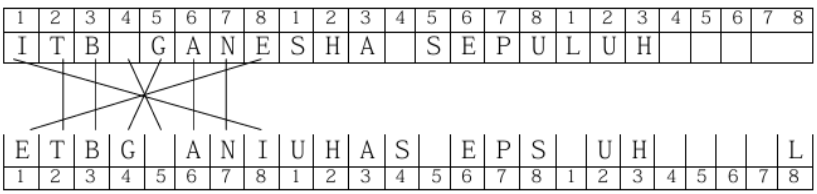
```
DEPARTEMEN TEKNIK INFORMATIKA ITB
```

Variasi dari metode transposisi lainnya ditunjukkan pada Contoh 5 dan Contoh 6.

Contoh 5. Misalkan plainteks adalah

```
ITB GANESHA SEPULUH
```

Plainteks diblok atas delapan karakter. Kemudian, pada tiap blok, karakter pertama dan karakter terakhir dipertukarkan, demikian juga karakter pertengahan:



maka cipherteksnya adalah

```
ETBG ANIUHAS EPS UH     L
```

Dekripsi dilakukan dengan cara yang sama, yaitu cipherteks diblok atas delapan karakter. Kemudian, pada tiap blok, karakter pertama dan karakter terakhir dipertukarkan, demikian juga karakter pertengahan.

Contoh 6. Misalkan plainteks adalah

```
CRYPTOGRAPHY AND DATA SECURITY
```

Plainteks disusun menjadi 3 baris (k = 3) seperti di bawah ini:

```
C     T     A     A     A     E     I
 R   P O   R P Y   N   D   T   S   C   R   T
   Y     G     H     D     A     U     Y
```

maka cipherteksnya adalah

```
CTAAAEIRPORPYNDTSCRTYGHDAUY
```

Kriptografi dengan alat *scytale* yang digunakan oleh tentara Sparta pada zaman Yunani termasuk ke dalam *cipher* transposisi.

4.4. Lebih Jauh dengan *Cipher* Abjad-tunggal

Seperti sudah disebutkan sebelum ini, metode *cipher* abjadtunggal mengganti setiap huruf di dalam abjad dengan sebuah huruf lain dalam abjad yang sama.

Jumlah kunci di dalam *cipher* abjad-tunggal sama dengan jumlah cara menyusun 26 huruf abjad tersebut, yaitu sebanyak

26! = 403.291.461.126.605.635.584.000.000

Ini berarti terdapat 26! buah kunci untuk menyusun hurufhuruf alfabet ke dalam tabel substitusi.

Contohnya, susunan huruf-huruf untuk cipherteks diperoleh dengan menyusun huruf-huruf abjad secara acak seperti tabel substitusi berikut:

Tabel substitusi:

p_i	:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
c_i	:	D	I	Q	M	T	B	Z	S	Y	K	V	O	F	E	R	J	A	U	W	P	X	H	L	C	N	G

Satu cara untuk membangkitkan kunci adalah dengan sebuah kalimat yang mudah diingat.

Misal kuncinya adalah

we hope you enjoy this book

Dari kunci tersebut, buang perulangan huruf sehingga menjadi

wehopyunjtisbk

lalu sambung dengan huruf-huruf lain yang tidak terdapat di dalam kalimat tersebut sehingga menjadi

W E H O P Y U N J T I S B K A C D F G L M Q R V X Z

Dengan demikian, tabel substitusi yang diperoleh adalah

Tabel substitusi:

p_i	:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
c_i	:	W	E	H	O	P	Y	U	N	J	T	I	S	B	K	A	C	D	F	G	L	M	Q	R	V	X	Z

4.5. Menerka Plainteks dari Cipherteks

Kadang-kadang kriptanalis melakukan terkaan untuk mengurangi jumlah kunci yang mungkin ada. Terkaan juga dilakukan kriptanalis untuk memperoleh sebanyak mungkin plainteks dari potongan cipherteks yang disadap. Plainteks yang diperoleh dari hasil terkaan ini biasanya digunakan dalam *known-plaintext attack*.

Asumsi yang digunakan: kriptanalis mengetahui bahwa pesan ditulis dalam Bahasa Inggris dan algoritma kriptografi yang digunakan adalah cipher abjad-tunggal

Contoh kasus 1: Kriptanalis mempunyai potongan cipherteks

G WR W RWL

Karena hanya ada dua kata yang panjangnya satu huruf dalam Bahasa Inggris (yaitu I dan A), maka **G** mungkin menyatakan huruf A dan **W** menyatakan huruf I, atau sebaliknya.

Kemungkinan **G** adalah huruf A dapat dieliminasi, maka diapstikan **G** = I, sehingga dengan cepat kriptanalis menyimpulkan bahwa potongan cipherteks tersebut adalah

I AM A MA*

Dengan pengetahuan Bahasa Inggris, karakter terakhir (*) hampir dipastikan adalah huruf N, sehingga kalimatnya menjadi

I AM A MAN

Hasil ini mengurangi jumlah kunci dari 26! menjadi 22!

Contoh kasus 2: Kriptanalis mempunyai potongan cipherteks

HKC

Tidak banyak informasi yang dapat disimpulkan dari *cryptogram* di atas. Namun kriptanalis dapat mengurangi beberapa kemungkinan kunci, karena sebagai contoh tidak mungkin Z diganti dengan **H**, Q dengan **K**, dan K dengan **C**.

Namun, jumlah kemungkinan kunci yang tersisa tetap masih besar. Jika pesan yang dikirim memang hanya tiga huruf, maka *exhaustive key search* akan menghasilkan kata dengan tiga huruf berbeda yang potensial sebagai plainteks.

Contoh kasus 3: Kriptanalis mempunyai potongan cipherteks

HATTPT

Dalam hal ini, kriptanalis dapat membatasi jumlah kemungkinan huruf plainteks yang dipetakan menjadi T.

Kriptanalis mungkin mendeduksi bahwa salah satu dari **T** atau **P** merepresentasikan huruf vokal. Kemungkinan plainteksnya adalah CHEESE, MISSES, dan CANNON.

Contoh kasus 4: Kriptanalis mempunyai potongan cipherteks

HATTPT

dan diketahui informasi bahwa pesan tersebut adalah nama negara. Dengan cepat kriptanalis menyimpulkan bahwa *polygram* tersebut adalah GREECE. Dalam hal ini, kriptanalis dapat membatasi jumlah kemungkinan huruf plainteks yang dipetakan menjadi T.

Kriptanalis mungkin mendeduksi bahwa salah satu dari **T** atau **P** merepresentasikan huruf vokal. Kemungkinan plainteksnya adalah CHEESE, MISSES, dan CANNON.

Metode Statistik dalam Kriptanalisis

Metode yang paling umum digunakan dalam memecahkan cipherteks adalah menggunakan statistik.

Dalam hal ini, kriptanalis menggunakan tabel frekuensi kemunculan huruf-huruf dalam teks bahasa Inggris. Tabel 4.2 memperlihatkan frekuensi kemunculan huruf-huruf abjad yang diambil dari sampel yang mencapai 300.000 karakter di dalam sejumlah novel dan suratkabar.

Tabel 4.2 Frekunsi kemunculan (relatif) huruf-huruf dalam teks Bahasa Inggris

Huruf	%	Huruf	%
A	8,2	N	6,7
B	1,5	O	7,5
C	2,8	P	1,9
D	4,2	Q	0,1
E	12,7	R	6,0
F	2,2	S	6,3
G	2,0	T	9,0
H	6,1	U	2,8
I	7,0	V	1,0
J	0,1	W	2,4
K	0,8	X	2,0
L	4,0	Y	0,1
M	2,4	Z	0,1

Tabel 4.2 di atas pada mulanya dipublikasikan di dalam *Cipher-Systems: The Protection of Communications* dan dikompilasi oleh H. J. Beker dan F.C. Piper

Terdapat sejumlah tabel frekuensi sejenis yang dipublikasikan oleh pengarang lain, namun secara umum persentase kemunculan tersebut konsisten pada sejumlah tabel.

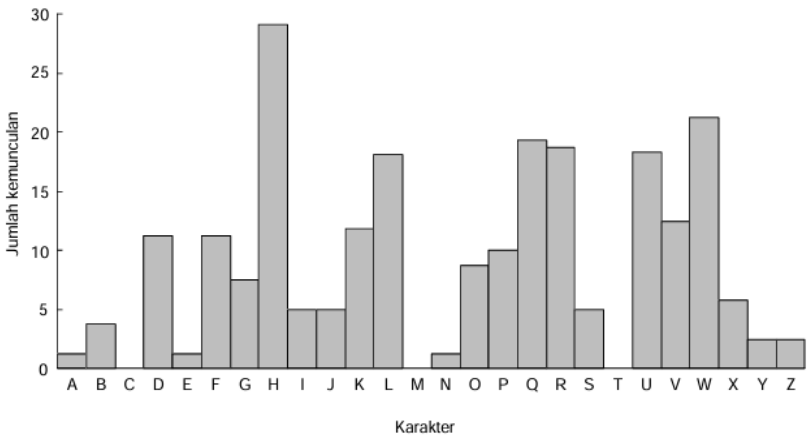
Bila *cipher* abjad tunggal digunakan untuk meng-engkripsi pesan, maka kemunculan huruf-huruf di dalam plainteks tercermin pada tabel 2 di atas. Misalnya bila di dalam *cipher* abjad tunggal huruf **R** menggantikan huruf E, maka frekuensi **R** di

dalam cipherteks relatif sama dengan frekuensi E di dalam plainteksnya.

Contoh 7.
Misalnya terdapat cipherteks yang panjang sebagai berikut:

```
DIX DR TZX KXCQDIQ RDK XIHPSZXKPIB TZPQ TXGT
PQ TD QZDM TZX KXCJXK ZDM XCQPVN TZPX TNSX DR
HPSZXK HCI LX LKDUXI. TZX MDKJ QTKFHTFKX DR
TZX SVCPITXGT ZCQ LXXI SKXQXKWXJ TD OCUX TZX
XGXKHPQX XCQPXK. PR MX ZCJ MKPTTXI TZX.
HKNSTDBKCOPI BKDFSQ DR RPWX VXTTXKQ TZXI PT
MDFVJ ZCWX LXXI ZCKJXK. TD HDIWPIHX
NDFKQXVWXQ DR TZPQ SCPKQ SCPKQ DR KXCJXKQ
HCI SKDWPJX XCHZ DTZXK MPTZ HKNSTDBKCOQ
MPTZ TZPQ VXTTXK BKDFSIB
```

Histogram yang memperlihatkan frekuensi kemunculan relatif huruf-huruf di dalam cipherteks diperlihatkan di bawah ini:



Gambar 4.1 Histogram yang menyatakan frekuensi kemunculan (relatif) huruf-huruf di dalam cipherteks di dalam Contoh 7

Dari histogram pada Gambar 4.1, karakter yang paling sering muncul di dalam cipherteks adalah **H**. Kita dapat menyimpulkan sementara bahwa **H** di dalam cipherteks menggantikan huruf E di dalam plainteks.

Cara yang sama dicoba untuk karakter-karakter lain di dalam cipherteks. Tetapi kita belum dapat memastikannya. Masih diperlukan:

- cara *trial and error*
- pengetahuan tentang bahasa
- konteks plainteks
- intuisi

5.
Cipher yang Tidak Dapat Dipecahkan (Unbreakable Cipher)

5.1. Pendahuluan

Unbreakable cipher merupakan klaim yang dibuat oleh kriptografer terhadap algoritma kriptografi yang dirancangnya.

Cipher substitusi (dengan segala varasinya) dan *cipher* transposisi sudah dibuktikan dapat dipecahkan. Kasus Queen Mary pada Abad 18 dan Enigma pada PD II adalah pelajaran betapa klaim *unbreakable cipher* mudah dipatahkan.

Apakah *unbreakable cipher* memang ada dan dapat dirancang? Jawabannya: ada dan bisa dibuat. Untuk merancang *unbreakable cipher*, ada dua syarat yang harus dipenuhi:

1. Kunci harus dipilih secara acak (yaitu, setiap kunci harus mempunyai peluang yang sama untuk terpilih).
2. Panjang kunci harus sama dengan panjang plainteks yang akan dienkripsikan.

Kedua syarat tersebut dapat menyebabkan *setiap* plainteks yang berbeda dan panjangnya sama akan sama-sama mempunyai kemungkinan menghasilkan cipherteks yang diberikan. Dengan kata lain, kriptanalis mendapatkan hasil bahwa cipherteks yang didekripsikannya menghasilkan beberapa plainteks yang mempunyai makna yang berbeda. Hal ini akan membingungkannya dalam menentukan plainteks yang benar.

Cipher yang tidak dapat dipecahkan dikatakan memiliki tingkat kerahasiaan yang sempurna (*perfect secrecy*). Satu-satunya algoritma kriptografi sempurna sehingga tidak dapat dipecahkan adalah *one-time pad*.

5.2. *One-Time Pad* (*OTP*)

OTP ditemukan pada tahun 1917 oleh Major Joseph Mauborgne. *Cipher* ini termasuk ke dalam kelompok algoritma kriptografi simetri. *One-time pad* (*pad* = kertas bloknot) berisi deretan karakterkarakter kunci yang dibangkitkan secara acak. Aslinya, satu buah *one-time pad* adalah sebuah pita (*tape*) yang berisi barisan karakter-karakter kunci.

Satu *pad* hanya digunakan sekali (*one-time*) saja untuk mengenkripsi pesan, setelah itu *pad* yang telah digunakan dihancurkan supaya tidak dipakai kembali untuk mengenkripsi pesan yang lain.

Aturan enkripsi yang digunakan persis sama seperti pada *Vigenere Cipher*. Pengirim pesan menggunakan setiap karakter kunci untuk mengenkripsikan satu karakter plainteks. Enkripsi dapat dinyatakan sebagai penjumlahan modulo 26 dari satu karakter plainteks dengan satu karakter kunci *onetime pad*:

$$c_i = (p_i + k_i) \bmod 26$$

Yang dalam hal ini,

p_i : karakter plainteks
k_i : karakter kunci
c_i : karakter cipherteks

Perhatikan bahwa panjang kunci sama dengan panjang plainteks, sehingga tidak ada kebutuhan mengulang penggunaan kunci selama proses enkripsi.

Setelah pengirim mengenkripsikan pesan dengan kunci, ia menghancurkan kunci tersebut (makanya disebut satu kali pakai atau *one-time*)

Penerima pesan menggunakan kunci yang sama untuk mendekripsikan karakter-karakter cipherteks menjadi karakter-karakter plainteks dengan persamaan:

$$p_i = (c_i - k_i) \bmod 26$$

Contoh 6.1: Misalkan plainteks dan kunci *one-time pad* adalah

plainteks: ONETIMEPAD
kunci: TBFRGFARFM

Nyatakan A = 0, B = 1, …, Z = 25, maka cipherteksnya adalah:

cipherteks: HOJKOREGHP

yang mana diperoleh sebagai berikut:

(O + T) mod 26 = H
(N + B) mod 26 = O
(E + F) mod 26 = J, dst

Sistem *OTP* ini tidak dapat dipecahkan karena:

1. Barisan kunci acak yang ditambahkan ke pesan plainteks yang tidak acak menghasilkan cipherteks yang seluruhnya acak.
2. Beberapa barisan kunci yang digunakan untuk mendekripsi cipherteks mungkin menghasilkan pesanpesan plainteks yang mempunyai makna, sehingga kriptanalis tidak punya cara untuk menentukan plainteks mana yang benar.

Contoh 6.2: Misalkan kriptanalis mencoba barisan kunci

LMCCAWAAZD

untuk mendekripsi cipherteks dari Contoh 5.1,

HOJKOREGHP

Plainteks yang dihasilkan (dengan menggunakan persamaan 5.2):

SALMONEGGS

Bila ia mencoba barisan kunci

ZDVUZOEYEO

maka plainteks yang dihasilkan adalah

GREENFIELD

Dua plainteks yang mempunyai makna ini membingungkan kriptanalis untuk memilih mana yang benar.

Dua plainteks yang mempunyai makna ini membingungkan kriptanalis untuk memilih mana yang benar.

5.3. Kelemahan OTP

Meskipun *One-Time Pad* (OTP) adalah sebuah cipher yang terbukti secara teoritis memiliki keamanan sempurna, fakta menunjukkan bahwa ia tidak digunakan secara universal dalam aplikasi kriptografi. Sebagai satu-satunya sistem cipher yang secara matematis tidak dapat dipecahkan, OTP menawarkan keamanan yang absolut jika aturan-aturan penggunaannya terpenuhi dengan ketat. Namun, dalam praktiknya, hanya sedikit sistem komunikasi yang menerapkan OTP. Sebaliknya, banyak aplikasi kriptografi masih menggunakan sistem cipher yang dapat dipecahkan atau yang memiliki tingkat kerentanan

tertentu, meskipun tetap dianggap cukup aman untuk penggunaan praktis.

Alasannya adalah dari segi kepraktisan, yaitu:

1. Karena panjang kunci harus sama dengan panjang pesan, maka *OTP* hanya cocok untuk pesan berukuran kecil. Semakin besar ukuran pesan, semakin besar pula ukuran kunci. Pada aplikasi kriptografi untuk mengenkripsikan data tersimpan, timbul masalah dalam penyimpanan kunci. Pada aplikasi kriptografi untuk komunikasi pesan, timbul masalah dalam pendistribusian kunci.
2. Karena kunci dibangkitkan secara acak, maka 'tidak mungkin' pengirim dan penerima membangkitkan kunci yang sama secara simultan. Jadi, salah seorang dari mereka harus membangkitkan kunci lalu mengirimkannya ke pihak lain.

Karena kerahasiaan kunci harus dijamin, maka perlu ada perlindungan selama pengiriman kunci. Jika hanya ada satu saluran komunikasi, maka pengirim dan penerima pesan perlu barisan kunci lain untuk melindungi kunci pertama, kunci ketiga untuk melindungi kunci kedua, dan seterusnya. Hal ini menghasilkan kumpulan barisan kunci yang tidak berhingga banyaknya.

Mengirimkan barisan kunci melalui saluran komunikasi yang digunakan untuk pengiriman pesan juga tidak praktis karena pertimbangan lalu lintas (*traffic*) pesan yang padat.

Oleh karena itu, *OTP* hanya dapat digunakan jika tersedia saluran komunikasi kedua yang cukup aman untuk mengirim kunci. Saluran kedua ini umumnya lambat dan mahal. Misalnya pada perang dingin antara AS dan Uni Soviet (dahulu), kunci

dibangkitkan, disimpan, lalu dikirim dengan menggunakan jasa kurir yang aman.

Penting diingat bahwa saluran kedua yang aman tersebut umumnya lambat dan mahal.

6.
Steganografi dan **Watermarking**

6.1. Definisi Steganografi

Steganografi adalah ilmu dan seni menyembunyikan pesan rahasia (*hiding message*) sehingga keberadaan pesan tersebut tidak dapat dideteksi oleh indera manusia. Istilah "steganografi" berasal dari bahasa Yunani yang berarti "tulisan tersembunyi" (*covered writing*). Dalam praktiknya, steganografi berfokus pada upaya membuat pesan rahasia menjadi tidak terlihat atau tidak terdengar, sehingga siapa pun yang tidak mengetahui rahasia tersebut akan menganggap media yang memuatnya sebagai data biasa tanpa ada informasi tambahan yang tersembunyi di dalamnya.

Untuk menyembunyikan informasi dengan steganografi, diperlukan dua elemen utama: wadah dan data rahasia. Wadah adalah media yang digunakan untuk menyimpan pesan tersembunyi, sedangkan data rahasia adalah pesan yang akan disembunyikan. Dalam *steganografi digital*, wadah biasanya berupa media digital, seperti gambar digital, suara, teks, atau video. Data rahasia yang disembunyikan juga dapat berupa berbagai jenis media digital, misalnya pesan teks yang disisipkan dalam gambar atau suara. Proses ini bekerja dengan mengubah atau menyisipkan data secara halus sehingga perubahan pada wadah tidak terlihat, dan orang yang melihat atau mendengarnya tidak akan menyadari pesan rahasia tersebut.

Steganografi sering dianggap sebagai kelanjutan dari kriptografi. Dalam kriptografi, data dikodekan menjadi bentuk

ciphertext yang, meskipun tidak dapat dibaca, masih terlihat dan menarik perhatian. Dalam steganografi, *ciphertext* dapat disembunyikan ke dalam wadah, sehingga tidak hanya isinya yang terlindungi, tetapi keberadaannya juga tidak diketahui. Hal ini menjadikan steganografi sebagai pendekatan keamanan ganda, karena data akan tampak sebagai bagian dari media digital biasa.

Di negara-negara yang memberlakukan penyensoran informasi yang ketat, steganografi sering digunakan untuk menyembunyikan pesan tertentu dari pengawasan otoritas. Misalnya, pesan yang dikodekan secara rahasia dapat disematkan ke dalam gambar, video, atau suara, dan kemudian dibagikan kepada publik tanpa dicurigai mengandung informasi terlarang. Penggunaan teknik steganografi memungkinkan pengguna untuk berkomunikasi secara rahasia, bahkan di bawah pengawasan ketat, tanpa mengungkapkan bahwa mereka sebenarnya sedang mengirim pesan.

6.2. Sejarah Steganografi

Steganografi sudah dikenal oleh bangsa Yunani. Herodatus, penguasa Yunani, mengirim pesan rahasia dengan menggunakan kepala budak atau prajurit sebagai media. Dalam hal ini, rambut budak dibotaki, lalu pesan rahasia ditulis pada kulit kepala budak. Ketika rambut budak tumbuh, budak tersebut diutus untuk membawa pesan rahasia di balik rambutnya.

Bangsa Romawi mengenal steganografi dengan menggunakan tinta tak tampak (*invisible ink*) untuk menuliskan pesan. Tinta tersebut dibuat dari campuran sari buah, susu, dan cuka. Jika tinta digunakan untuk menulis maka tulisannya tidak tampak. Tulisan di atas kertas dapat dibaca dengan cara memanaskan kertas tersebut.

Sebagai contoh ilustrasi, di bawah ini adalah citra lada (peppers.bmp) yang akan digunakan untuk menyembunyikan sebuah dokumen *word* (hendro.doc).

Gambar 6.1 *Peppers.bmp*

LETTER OF RECOMMENDATION

To Whom It May Concern,

Herewith I highly recommend **Mr. R. Hendro Wicaksono** continue his postgradu study at your university. My recommendation is based on my experience as lecturer in several courses for the past four years.

He has shown me his excellent attitude and personality. He is a hard working pers and he has a lot of creative ideas. He is also a very intelligent student and cooperates very well with his peers whenever they had to work together.

During his study, he showed diligence and eagerness to achieve his goal. He sets v high standard for himself and organizes himself very well to achieve the standar am confident that if he can maintain his goal work, he should be able to complete postgraduate program well within the stipulated time.

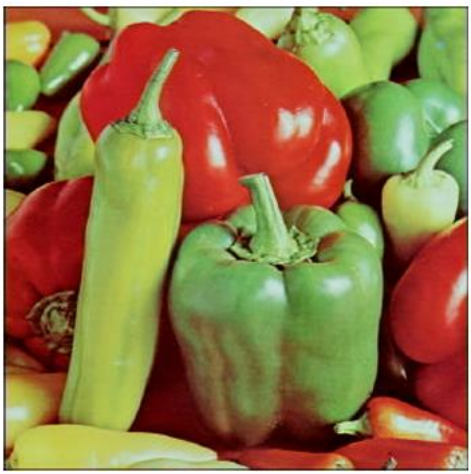
I am sure that his abilities and his personal qualities along with his acade capabilities will help his to obtain his Master's degree at your university, which w be very useful for our country.

Bandung, November 15, 2002
Yours Sincerely,

Ir. Rinaldi Munir, M.Sc.
Senior Lecturer
Informatics Engineering Department,
Institute Technology of Bandung (ITB)
Jl. Ganesha No. 10, Bandung 40132
Email : rinaldi@informatika.org
Phone +62-22-2508135
Indonesia

Gambar 6.2 hendro.doc

Hasil penyembunyian data (peppers.bmp + hendro.doc)



Gambar 6.3 Citra lada setelah "diisi" dengan data teks.

6.3. Kriteria Steganografi yang Bagus

Steganografi yang dibahas di sini adalah penyembunyian data di dalam citra digital saja. Meskipun demikian, penyembunyian data dapat juga dilakukan pada wadah berupa suara digital, teks, ataupun video.

Penyembunyian data rahasia ke dalam citra digital akan mengubah kualitas citra tersebut. Kriteria yang harus diperhatikan dalam penyembunyian data adalah:

1. *Fidelity*. Mutu citra penampung tidak jauh berubah. Setelah penambahan data rahasia, citra hasil steganografi masih terlihat dengan baik. Pengamat tidak mengetahui kalau di dalam citra tersebut terdapat data rahasia.
2. *Robustness*. Data yang disembunyikan harus tahan terhadap manipulasi yang dilakukan pada citra penampung (seperti pengubahan kontras, penajaman, pemampatan, rotasi, perbesaran gambar, pemotongan (*cropping*), enkripsi, dan sebagainya). Bila pada citra dilakukan operasi pengolahan citra, maka data yang disembunyikan tidak rusak.

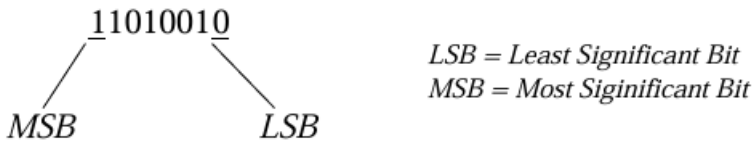
3. *Recovery*. Data yang disembunyikan harus dapat diungkapkan kembali (*recovery*). Karena tujuan steganografi adalah *data hiding*, maka sewaktu-waktu data rahasia di dalam citra penampung harus dapat diambil kembali untuk digunakan lebih lanjut.

6.4. Teknik Penyembunyian Data

Penyembunyian data dilakukan dengan mengganti bit-bit data di dalam segmen citra dengan bit-bit data rahasia. Metode yang paling sederhana adalah metode **modifikasi *LSB*** (*Least Significant Bit Modification*).

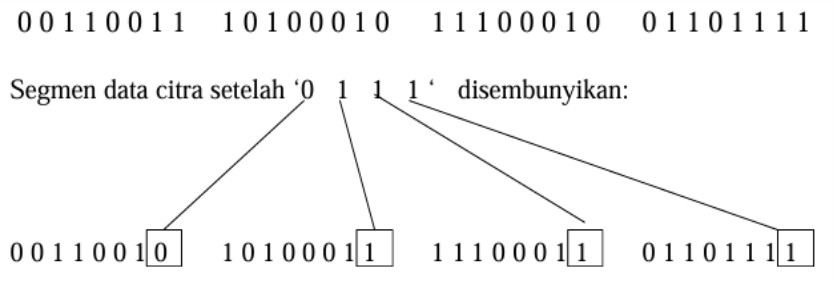
Pada susunan bit di dalam sebuah *byte* (1 *byte* = 8 bit), ada bit yang paling berarti (*most significant bit* atau *MSB*) dan bit yang paling kurang berarti (*least significant bit* atau *LSB*).

Perhatikan contoh sebuah susunan bit pada sebuah *byte*:



Bit yang cocok untuk diganti adalah bit *LSB*, sebab perubahan tersebut hanya mengubah nilai *byte* satu lebih tinggi atau satu lebih rendah dari nilai sebelumnya. Misalkan *byte* tersebut menyatakan warna merah, maka perubahan satu bit LSB tidak mengubah warna merah tersebut secara berarti. Lagi pula, mata manusia tidak dapat membedakan perubahan yang kecil.

Misalkan segmen data citra sebelum perubahan:



Untuk memperkuat teknik penyembunyian data, bit-bit data rahasia tidak digunakan mengganti *byte-byte* yang berurutan, namun dipilih susunan *byte* secara acak. Misalnya jika terdapat 50 *byte* dan 6 bit data yang akan disembunyikan, maka maka *byte* yang diganti bit *LSB*-nya dipilih secara acak, misalkan *byte* nomor 36, 5, 21, 10, 18, 49.

Bilangan acak dapat dibangkitkan dengan program *pseudorandom-number-generator* (*PRNG*). *PRNG* menggunakan kunci rahasia untuk membangkitkan posisi *pixel* yang akan digunakan untuk menyembunyikan bit-bit.

PRNG dibangun dalam sejumlah cara, salah satunya dengan menggunakan algoritma kriptografi berbasis blok (*block cipher*). Tujuan dari enkripsi adalah menghasilkan sekumpulan bilangan acak yang sama untuk setiap kunci enkripsi yang sama. Bilangan acak dihasilkan dengan cara memilih bit-bit dari sebuah blok data hasil enkripsi.

Sayangnya, metode modifikasi *LSB* kurang bagus untuk steganografi, karena *robustness*-nya rendah. Selain itu, dapat terjadi kasus penurunan jumlah warna (*fidelity* rendah).

6.5. Ukuran Data Yang Disembunyikan

Ukuran data yang akan disembunyikan bergantung pada ukuran citra penampung. Pada citra 24-bit yang berukuran 256 × 256 *pixel* terdapat 65536 *pixel*, setiap *pixel* berukuran 3 *byte*

(komponen *RGB*), berarti seluruhnya ada 65536 × 3 = 196608 *byte*. Karena setiap *byte* hanya bisa menyembunyikan satu bit di *LSB*-nya, maka ukuran data yang akan disembunyikan di dalam citra maksimum 196608/8 = 24576 *byte*

Ukuran data ini harus dikurangi dengan panjang nama berkas, karena penyembunyian data rahasia tidak hanya menyembunyikan isi data tersebut, tetapi juga nama berkasnya.

Semakin besar data disembunyikan di dalam citra, semakin besar pula kemungkinan data tersebut rusak akibat manipulasi pada citra penampung.

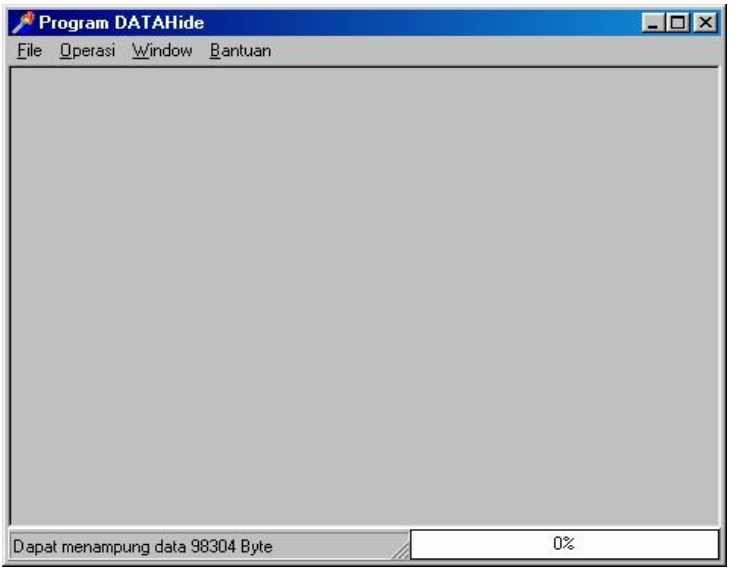
Untuk memperkuat keamanan, data yang akan disembunyikan dapat dienkripsi terlebih dahulu. Sedangkan untuk memperkecil ukuran data, data dimampatkan sebelum disembunyikan. Bahkan, pemampatan dan enkripsi dapat juga dikombinasikan sebelum melakukan penyembunyian data.

6.6. Teknik Ektsraksi Data

Data yang disembunyikan di dalam citra dapat dibaca kembali dengan cara pengungkapan (reveal atau extraction). Posisi byte yang menyimpan bit data dapat diketahui dari bilangan acak yang dibangkitkan. Bilangan acak yang dihasilkan harus sama dengan bilangan acak yang dipakai pada waktu penyembunyian data. Dengan demikian, bit-bit data rahasia yang bertaburan di dalam citra dapat dikumpulkan kembali.

Contoh program steganografi adalah DATAhide (dari program Tugas Akhir Lazarus Poli, NIM : 13593601, Penerapan Steganografi dengan Citra Dijital Sebagai File Penampung, Tugas Akhir Teknik Informatika, 1998).

Untuk setiap contoh, digunakan kunci enkripsi yang sama: informatika



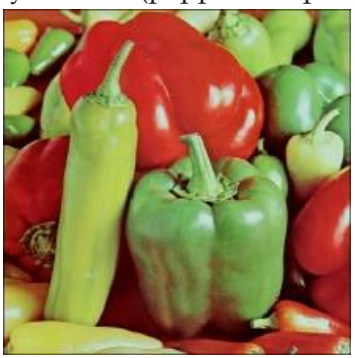
Upa-menu pada Operasi: Penyembunyian data dan Pengungkapan data.

Citra 24 bit

Penampung: citra peppers.bmp (512 × 512 *pixel*, 769 KB)

Data yang disembunyikan: citra handshak.bmp (44 KB)

Hasil penyembunyian data (peppers.bmp + handshake.bmp):



Hasil ekstraksi data:

Berkas handshak-stega.bmp hasil ekstraksi:

Citra penampung: pepper.bmp (512 × 512 pixel), lihat Gambar 6.1.

SData yang disembunyikan: citra hendro.doc (20 KB)

LETTER OF RECOMMENDATION

To Whom It May Concern,

Herewith I highly recommend **Mr. R. Hendro Wicaksono** continue his postgradua
study at your university. My recommendation is based on my experience as his lecturer
several courses for the past four years.

He has shown me his excellent attitude and personality. He is a hard working person a
he has a lot of creative ideas. He is also a very intelligent student and he cooperates ve
well with his peers whenever they had to work together.

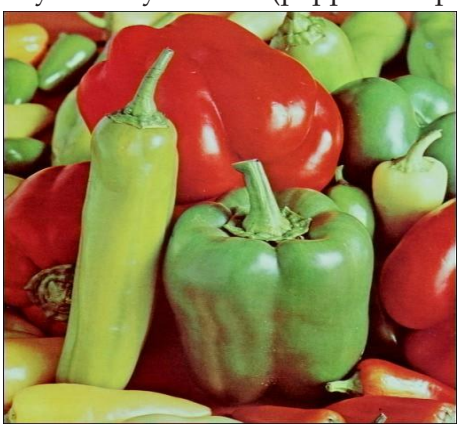
During his study, he showed diligence and eagerness to achieve his goal. He sets ve
high standard for himself and organizes himself very well to achieve the standard. I a
confident that if he can maintain his goal work, he should be able to complete
postgraduate program well within the stipulated time.

I am sure that his abilities and his personal qualities along with his academic capabilit
will help his to obtain his Master's degree at your university, which will be very use
for our country.

Bandung, November 15, 2002
Yours Sincerely,

Ir. Rinaldi Munir, M.Sc.
Senior Lecturer
Informatics Engineering Department,
Institute Technology of Bandung (ITB)
Jl. Ganesha No. 10, Bandung 40132
Email : rinaldi@informatika.org
Phone +62-22-2508135
Indonesia

Hasil penyembunyian data (peppers.bmp + hendro.doc)



Hasil ekstraksi data:

Berkas hendro-stega.doc hasil ekstraksi:

LETTER OF RECOMMENDATION

To Whom It May Concern,

Herewith I highly recommend **Mr. R. Hendro Wicaksono** continue his postgradua study at your university. My recommendation is based on my experience as his lecturer several courses for the past four years.

He has shown me his excellent attitude and personality. He is a hard working person a he has a lot of creative ideas. He is also a very intelligent student and he cooperates ve well with his peers whenever they had to work together.

During his study, he showed diligence and eagerness to achieve his goal. He sets ve high standard for himself and organizes himself very well to achieve the standard. I a confident that if he can maintain his goal work, he should be able to complete t postgraduate program well within the stipulated time.

I am sure that his abilities and his personal qualities along with his academic capabilit will help his to obtain his Master's degree at your university, which will be very use for our country.

Bandung, November 15, 2002
Yours Sincerely,

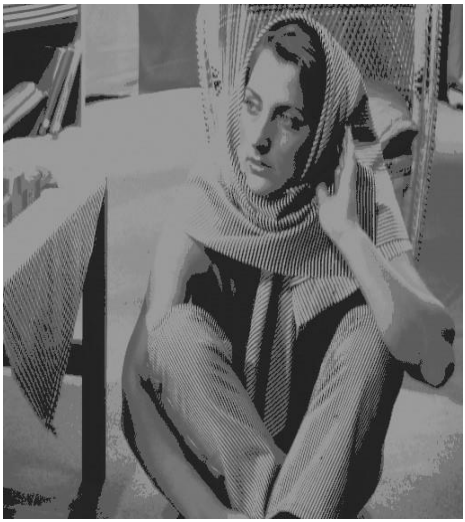
Ir. Rinaldi Munir, M.Sc.
Senior Lecturer
Informatics Engineering Department,
Institute Technology of Bandung (ITB)
Jl. Ganesha No. 10, Bandung 40132
Email : rinaldi@informatika.org
Phone +62-22-2508135
Indonesia

Citra 8 bit

Penampung: citra barbara.bmp (512 × 512 *pixel*, 258 KB)

Data yang disembunyikan: chord.wav (95 KB) – berkas musik dari Windows.

Hasil penyembunyian data (barbara.bmp + chord.wav)



Terjadi penurunan kualitas gambar karena pengaruh penurunan jumlah warna (*color quantization*)!

Hasil ektraksi data:

6.7. Watermarking

Salah satu karya intelektual yang dilindungi adalah produk dalam bentuk digital, seperti software dan produk multimedia seperti teks, musik (dalam format MP3 atau WAV), gambar/citra (image), dan video digital (VCD). Selama ini penggandaan atas produk digital tersebut dilakukan secara bebas dan leluasa. Pemegang hak cipta atas produk digital tersebut tentu dirugikan karena ia tidak mendapat royalti dari usaha penggandaan tersebut.

Salah satu cara untuk melindungi hak milik intelektual atas produk multimedia (gambar/foto, audio, teks, video) adalah dengan menyisipkan informasi ke dalam data multimedia tersebut dengan teknik digital watermarking. Informasi yang disisipkan ke dalam data multimedia disebut watermark, dan watermark dapat dianggap sebagai sidik digital (digital signature) atau stempel digital dari pemilik yang sah atas produk multimedia tersebut.

Pemberian signature dengan teknik watermarking ini dilakukan sedemikian sehingga informasi yang disisipkan tidak merusak data digital yang dilindungi.

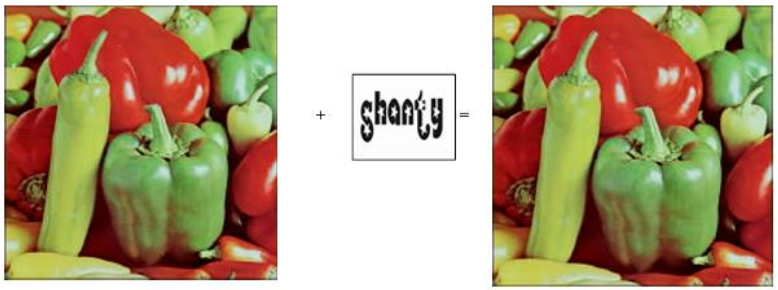
Watermark di dalam data digital tidak dapat dideteksi oleh orang yang tidak mengetahui rahasia skema penyisipan watermark, dan juga watermark tidak dapat diidentifikasi dan dihilangkan.

Watermark dapat digunakan sebagai bukti kepemilikan untuk membantu digital publisher melindungi materi yang mempunyai hak cipta (copyright).

Jika ada orang lain yang mengklaim bahwa produk digital yang didapatkannya adalah miliknya, maka pemegang hak cipta atas karya multimedia tersebut dapat membantah klaim tersebut dengan proses verifikasi. Caranya, watermark diekstraksi dari produk digital yang disengketakan. Watermark yang diekstraksi tersebut dibandingkan dengan watermark pemegang hak cipta. Jika sama, berarti memang dialah pemegang hak cipta produk multimedia tersebut.

Pada dasarnya, teknik watermarking adalah proses menambahkan kode identifikasi secara permanen ke dalam data digital. Kode identifikasi tersebut dapat berupa teks, gambar, suara, atau video. Selain tidak merusak data digital produk yang akan dilindungi, kode yang disisipkan seharusnya memiliki ketahanan (robustness) dari berbagai pemrosesan lanjutan seperti pengubahan, transformasi geometri, kompresi, enkripsi, dan sebagainya. Sifat robustness berarti data watermark tidak rusak akibat pemrosesan lanjutan tersebut.

Gambar 6.4 memperlihatkan sebuah gambar (image) paprika yang disisipi dengan watermark berupa gambar hitam putih yang menyatakan identifikasi pemiliknya (Shanty). Perhatikanlah bahwa setelah disisipi watermark, gambar paprika tetap kelihatan mulus, seolah-olah tidak pernah disisipi watermark sebelumnya. Sebenarnya tidaklah demikian, gambar paprika tersebut mengalami sedikit perubahan akibat watermarking, namun mata manusia mempunyai sifat kurang peka terhadap perubahan kecil ini, sehingga manusia sukar membedakan mana gambar yang asli dan mana gambar yang sudah disisipi *watermark*.



Gambar 6.4. Memberi watermark pada citra peppers

(*output* program Tugas Akhir Shanty Meliani H., 13599059, *Robust and Non Blind Watermarking pada Citra Dijital dengan Teknik Spread Spectrum*)

Berdasarkan tipe dokumen yang diberi watermark, watermarking dapat diklasifikasikan menjadi:

1. Image Watermarking
2. Video Watermarking
3. Audio Watermarking
4. Text Watermarking

Watermarking dapat juga dikategorikan sebagai visible watermarking (watermark terlihat oleh indera manusia) dan invisible watermarking (watermark tidak tampak).

Watermarking dapat juga dikategorikan sebagai blind watermarking (proses verifikasi watermark tidak membutuhkan citra asal) dan non blind watermarking (proses verifikasi watermark membutuhkan citra asal).

6.8. Sejarah Watermarking

Watermarking sudah ada sejak 700 tahun yang lalu. Pada akhir abad 13, pabrik kertas di Fabriano, Italia, membuat kertas yang diberi watermark atau tanda-air dengan cara menekan

bentuk cetakan gambar atau tulisan pada kertas yang baru setengah jadi. Ketika kertas dikeringkan terbentuklah suatu kertas yang ber-*watermark*. Kertas ini biasanya digunakan oleh seniman atau sastrawan untuk menulis karya mereka. Kertas yang sudah dibubuhi tanda-air tersebut sekalius dijadikan identifikasi bahwa karya seni di atasnya adalah milik mereka.

Ide *watermarking* pada data digital (sehingga disebut digital watermarking) dikembangkan di Jepang tahun 1990 dan di Swiss tahun 1993. Digital watermarking semakin berkembang seiring dengan semakin meluasnya penggunaan internet, objek digital seperti video, citra, dan suara yang dapat dengan mudah digandakan dan disebarluaskan.

6.9. Penyisipan *Watermark*

Watermarking merupakan aplikasi dari steganografi, namun ada perbedaan antara keduanya. Jika pada steganografi informasi rahasia disembunyikan di dalam media digital dimana media penampung tidak berarti apa-apa, maka pada watermarking justru media digital tersebut yang akan dilindungi kepemilikannya dengan pemberian label hak cipta.

Meskipun steganografi dan watermarking tidak sama, namun secara prinsip proses penyisipan informasi ke dalam data digital tidak jauh berbeda.

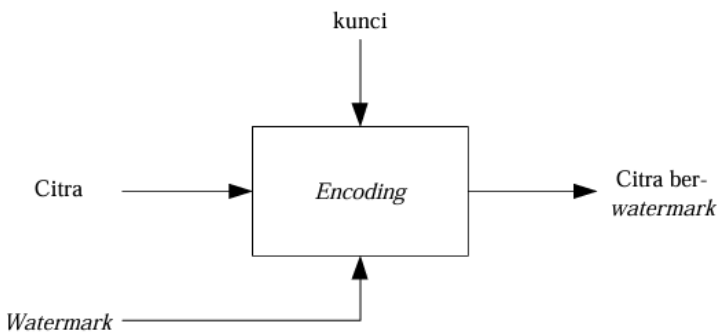
Data watermark yang lazim disisipkan ke dalam data digital adalah teks, citra, atau suara. Watermark berupa teks misalnya pernyataan atau pesan yang mengindikasikan kepemilikan dokumen (copyright notification). Watermark berupa teks mengandung kelemahan karena kesalahan satu bit akan menghasilkan hasil teks yang berbeda pada waktu verifikasi (ektraksi).

Watermark berupa suara atau citra lebih disukai karena kesalahan pada beberapa bit *watermark* tidak menghasilkan

perubahan yang berarti pada waktu verifikasi. Hasil ekstraksi watermark yang mengandung kesalahan tersebut masih dapat dipersepsi secara visual (atau secara pendengaran jika watermark-nya berupa suara). Citra yang sering digunakan sebagai watermark biasanya logo perusahaan atau lambang.

6.10. Teknik Penyembunyian Data

Di sini kita hanya meninjau watermarking pada citra digital. Proses penyisipan *watermark* ke dalam citra disebut encoding dan ditunjukkan Gambar 7.5. Encoding dapat disertai dengan pemasukan kunci atau tidak memerlukan kunci. Kunci diperlukan agar watermark hanya dapat diekstraksi oleh pihak yang sah. Kunci juga dimaksudkan untuk mencegah watermark dihapus oleh pihak yang tidak berhak.



Gambar 6.5 Proses penyisipan watermark pada citra digital

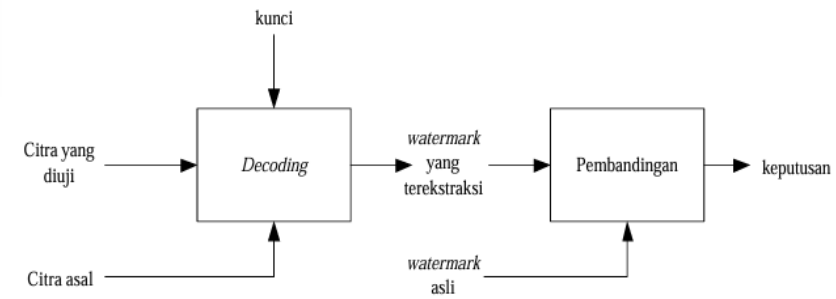
6.11. Verifikasi Watermark

Verifikasi watermark dilakukan untuk membuktikan status kepemilikan citra digital yang disengketakan. Verifikasi watermark terdiri atas dua sub-proses, yaitu ekstraksi watermark dan pembandingan.

Sub-proses ekstraksi *watermark* disebut juga decoding, bertujuan mengungkap *watermark* dari dalam citra. Decoding dapat mengikutsertakan citra asal (yang belum diberi watermark)

atau tidak sama sekali, karena beberapa skema watermarking memang menggunakan citra asal dalam proses decoding untuk meningkatkan unjuk kerja yang lebih baik [HEN03].

Sub-proses pembandingan bertujuan membandingkan watermark yang diungkap dengan watermark asli dan memberi keputusan tentang watermark tersebut. Proses verifikasi watermark ditunjukkan pada Gambar 7.6.



Gambar 6.6 Proses verifikasi watermark pada citra digital

6.12. Tujuan Watermarking Lainnya

Selain untuk tujuan pelabelan hak cipta (copyright labelling), watermarking juga dimanfaatkan untuk tujuan-tujuan lain sebagai berikut [SUP00]:

1. *Tamper-proofing*. Watermarking digunakan sebagai alat untuk mengidentifikasi atau menunjukkan bahwa data digital telah mengalami perubahan dari aslinya.
2. *Feature location*. Watermarking digunakan untuk mengidentifikasi isi dari data digital pada lokasi-lokasi tertentu.
3. *Annotation/caption*. Watermarking digunakan hanya sebagai keterangan tentang data digital itu sendiri.

6.13. *Watermarking* pada Media Digital Lain

Sebagian besar penelitian, publikasi, dan aplikasi di bidang watermarking ditujuan untuk citra digital. Namun, *watermarking* juga dapat diterapkan pada jenis multimedia lain seperti suara (misalnya musik MP3), video, dan teks.

Sedangkan watermarking pada video digital harus sedemikian rupa sehingga peralihan gambar dari satu frame ke frame lainnya harus tetap baik dan tidak terlihat dimodifikasi. Karena video digital ukurannya relatif besar daripada citra, maka *watermark* yang disisipkan dapat lebih banyak.

Khusus watermarking pada data audio, kehati-hatian perlu dilakukan pada perancangan algoritma *watermarking*-nya, karena suara lebih sensitif daripada gambar. Hal ini berarti suara digital lebih mudah rusak bila ditambahkan watermarking.

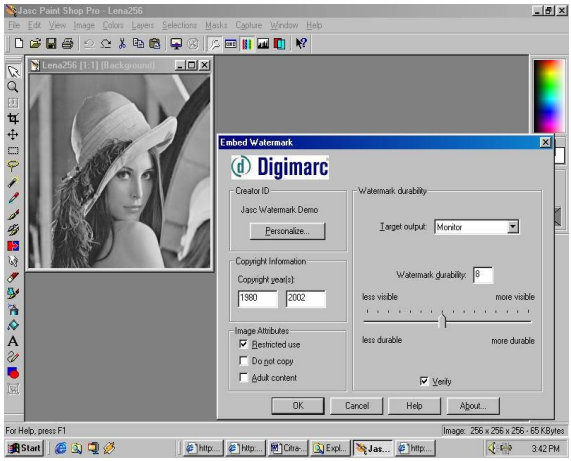
Saat ini, Microsoft sedang meneliti untuk mengembangkan sistem *watermarking* untuk audio digital, yang akan dimasukkan ke dalam media player Windows. Dengan sistem *watermarking* ini, data seperti informasi lisensi disisipkan ke dalam musik/lagu; media player tidak akan memainkan file audio yang memuat *watermark* yang salah.

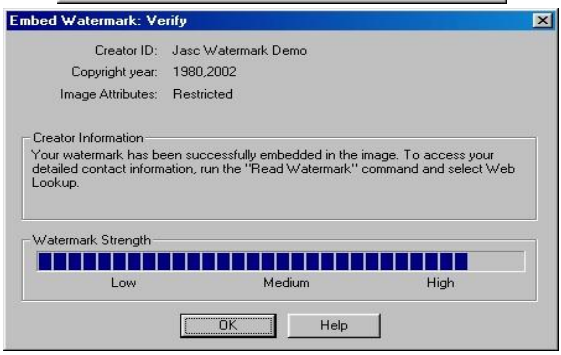
Terakhir, *watermarking* pada dokumen teks menggunakan metode yang berbeda daripada 3 media lainnya. Salah satunya dengan menyisipkan spasi antara dua buah kata atau antara dua buah kalimat di dalam dokumen.

6.14. *Watermarking* pada Program Komersil

Banyak program multimedia komersil dilengkapi dengan menu untuk menambahkan watermark, seperti pada Adobe Photoshop 5.5 (www.adobe.com) dan PaintShop Pro. 6.

Contoh menu *watermark* pada program Paintshop Pro. 6:





Citra Lena sesudah pengisian informasi *watermark*:

Digital *watermarking* adalah teknologi yang memungkinkan penyisipan tanda pengenal unik atau *watermark* dalam media digital, seperti musik dan gambar, untuk melindungi hak cipta dan melacak distribusi konten. Salah satu perangkat lunak digital watermarking adalah **Giovanni™** dari Blue Spike (tersedia di), yang menggunakan kunci kriptografi untuk menghasilkan watermark pada musik dan citra digital, menjadikannya aman dari upaya penghapusan. Beberapa perusahaan software lainnya yang menawarkan solusi dalam bidang ini, seperti **Digimarc** (tersedia di www.digimarc.com) dan **Cognicity** (di www.cognicity.com), juga memiliki teknologi dan layanan dalam digital watermarking.

Berikut adalah beberapa situs yang menyediakan informasi tambahan dan perangkat lunak:

1. (www.outguess.org) Situs ini menawarkan alat steganografi gratis untuk menyembunyikan informasi dalam gambar.
2. (www.demcom.com) Perangkat lunak ini memungkinkan pengguna untuk mengenkripsi serta menyembunyikan file dalam format audio, video, teks, atau HTML, memberikan lapisan keamanan tambahan.

3. (www.cl.cam.ac.uk/~fapp2/steganography/index.html) - Halaman ini menyediakan informasi teknis, berita, dan tautan terkait steganografi serta digital watermarking.
4. (www.digimarc.com) Selain homepage utama yang menawarkan solusi watermarking, situs ini berfungsi sebagai pusat informasi terkait produk-produk mereka untuk keamanan dan pelacakan konten digital.

7.
Tipe dan Mode **Algoritma Simetri**

7.1. Pendahuluan

Algoritma kriptografi (cipher) yang beroperasi dalam mode bit dapat dikelompokkan menjadi dua kategori:

1. Cipher aliran (stream cipher) Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal, yang dalam hal ini rangkaian bit dienkripsikan/didekripsikan bit per bit.
2. Cipher blok (block cipher) Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk blok bit, yang dalam hal ini rangkaian bit dibagi menjadi blok-blok bit yang panjangnya sudah ditentukan sebelumnya. Misalnya panjang blok adalah 64 bit, maka itu berarti algoritma enkripsi memperlakukan 8 karakter setiap kali penyandian (1 karakter = 8 bit dalam pengkodean ASCII).

Baik cipher aliran maupun cipher blok, keduanya termasuk ke dalam algoritma kriptografi simetri.

7.2. Aliran

Chiper aliran mengenkripsikan plainteks menjadi chiperteks bit per bit (1 bit setiap kali transformasi).

Catatan: Variasi chiper aliran lainnya adalah mengenkripsikan plainteks menjadi chiperteks karakter per karakter atau kata per kata, misalnya pada Vigenere Cipher dan one-time pad chiper.

Cipher aliran pertama kali diperkenalkan oleh Vernam melalui algoritmanya yang dikenal dengan nama **Vernam**.

Vernam cipher diadopsi dari one-time pad cipher, yang dalam hal ini karakter diganti dengan bit (0 atau 1). Cipherteks diperoleh dengan melakukan penjumlahan modulo 2 satu bit plainteks dengan satu bit kunci:

$$c_i = (p_i + k_i) \bmod 2 \qquad \textbf{(7.1)}$$

yang dalam hal ini,

p_i : bit plainteks
k_i : bit kunci
c_i : bit cipherteks

Plainteks diperoleh dengan melakukan penjumlahan modulo 2 satu bit cipherteks dengan satu bit kunci:

$$p_i = (c_i - k_i) \bmod 2 \qquad \textbf{(7.2)}$$

(perhatikan bahwa untuk sistem biner, persamaan 7.2 identik dengan $p_i = (c_i + k_i) \bmod 2$)
Dengan kata lain, Vernam *cipher* adalah versi lain dari *onetime pad cipher*. Oleh karena operasi penjumlahan modulo 2 identik dengan operasi bit dengan operator XOR, maka persaman (6.3) dapat ditulis sebagai

$$c_i = p_i \quad k_i \qquad \textbf{(7.3)}$$

dan proses deksripsi menggunakan persamaan

$$p_i = c_i \quad k_i \qquad \textbf{(7.4)}$$

Pada *cipher* aliran, bit hanya mempunyai dua buah nilai, sehingga proses enkripsi hanya menyebabkan dua keadaan pada bit tersebut: berubah atau tidak berubah. Dua keadaan tersebut ditentukan oleh kunci enkripsi yang disebut **aliranbit-kunci** (*keystream*).

Aliran-bit-kunci dibangkitkan dari sebuah pembangkit yang dinamakan pembangkit aliran-bit-kunci (*keystream generator*). Aliran-bit-kunci (sering dinamakan *running key*) di-*XOR*-kan dengan aliran bit-bit plainteks, p_1, p_2, …, p_i, untuk menghasilkan aliran bit-bit cipherteks:

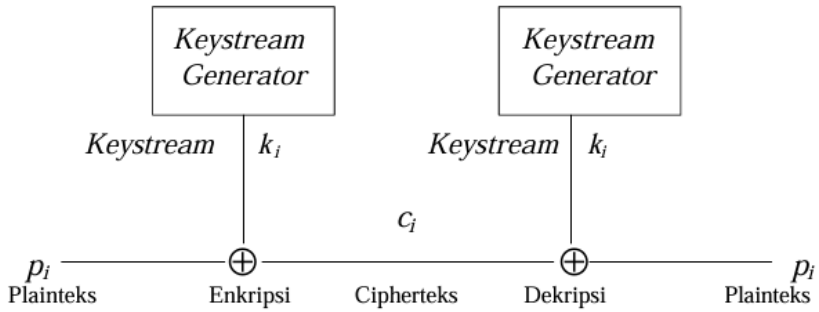
$$p_i = c_i \quad k_i$$

karena

$$c_i \quad k_i = (p_i \quad k_i) \quad k_i = p_i \quad (k_i \quad k_i) = p_i \quad 0 = p_i$$

Catatlah bahwa proses enkripsi dua kali berturut-turut menghasilkan kembali plainteks semula.

Gambar 7.1 memperlihatkan konsep *cipher* aliran.



Gambar 7.1 Konsep *cipher* aliran

Contoh 7.1: Misalkan plainteks adalah

1100101

dan aliran-bit-kunci adalah

1000110

maka cipherteks yang dihasilkan adalah

0100011

yang mana diperoleh dengan meng-XOR-kan bit-bit plainteks dengan bit-bit aliran-kunci pada posisi yang berkoresponden.

Keamanan sistem cipher aliran bergantung seluruhnya pada pembangkit aliran-bit-kunci. Jika pembangkit mengeluarkan aliran-bit-kunci yang seluruhnya nol, maka cipherteks sama dengan plainteks, dan proses enkripsi menjadi tidak artinya.

Jika pembangkit mengeluarkan aliran-bit-kunci dengan pola 16-bit yang berulang, maka algoritma enkripsinya menjadi sama seperti enkripsi dengan XOR sederhana yang memiliki tingkat keamanan yang tidak berarti.

Jika pembangkit mengeluarkan aliran-bit-kunci yang benar benar acak (truly random), maka algoritma enkripsinya sama dengan *one-time* pad dengan tingkat keamanan yang sempurna. Pada kasus ini, aliran-bit-kunci sama panjangnya dengan panjang plainteks, dan kita mendapatkan cipher aliran sebagai unbreakable cipher.

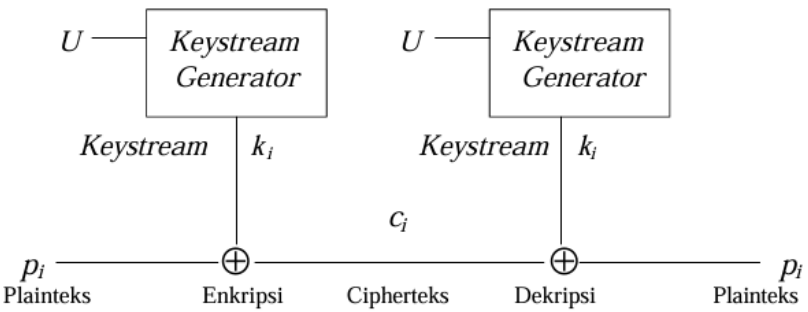
Tingkat keamanan cipher aliran terletak antara algoritma XOR sederhana dengan *one-time* pad. Semakin acak keluaran yang dihasilkan oleh pembangkit aliran-bit-kunci, semakin sulit kriptanalis memecahkan cipherteks.

(Keystream Generator)

Pembangkit aliran-bit-kunci dapat membangkitkan bit-bit kunci (keystream) berbasis bit per bit atau dalam bentuk blok-blok bit. Untuk yang terakhir ini, cipher blok dapat digunakan untuk untuk memperoleh cipher aliran.

Untuk alasan praktis, pembangkit aliran-bit kunci diimplementasikan sebagai prosedur algoritmik, sehingga bit bit kunci (keystream) dapat dibangkitkan secara simultan oleh pengirim dan penerima pesan.

Prosedur algoritmik tersebut menerima masukan sebuah kunci U. Keluaran dari prosedur merupakan fungsi dari U (lihat Gambar 7.2). Pembangkit harus menghasilkan bit-bit kunci yang kuat secara kriptografi.



Gambar 7.2 Cipher aliran dengan pembangkit aliran-bit-kunci yang bergantung pada kunci U.

Karena pengirim dan penerima harus menghasilkan bit-bit kunci yang sama , maka keduanya harus memiliki kunci U yang sama. Kunci U ini harus dijaga kerahasiaanya.

Cipher aliran menggunakan kunci U yang relatif pendek untuk membangkitkan bit-bit kunci yang panjang.

Contoh 7.2: Misalkan U adalah kunci empat-bit yang dipilih sembarang, kecuali 0000. Bit-aliran-kunci yang dibangkitkan akan berulang setiap 15 bit. Misalkan

$U = 1111$

Barisan bit-bit kunci diperoleh dengan meng-XOR-kan bit pertama dengan bit terakhir dari empat bit sebelumnya, sehingga menghasilkan:

111101011001000

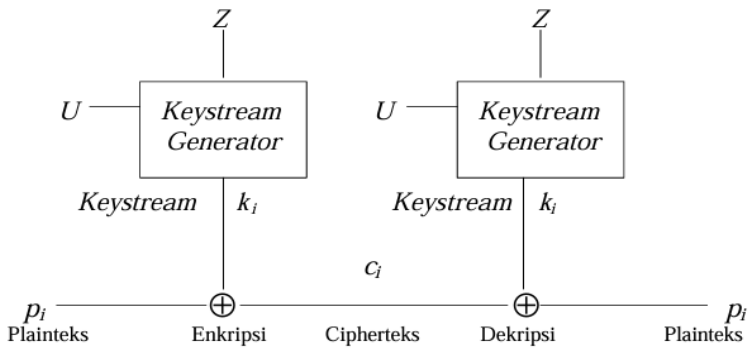
dan akan berulang setiap 15 bit.

Secara umum, jika panjang kunci U adalah n bit, maka bit-bit kunci tidak akan berulang sampai 2n – 1 bit.

Karena U adalah besaran yang konstan, maka aliran bit-bit kunci yang dihasilkan pada setiap lelaran tidak berubah jika bergantung hanya pada U.

Ini berarti pembangkit aliran-bit-kunci tidak boleh mulai dengan kondisi awal yang sama supaya tidak menghasilkan kembali bit-bit kunci yang sama pada setiap lelaran.

Oleh karena itu, beberapa pembangkit aliran-bit-kunci menggunakan umpan (seed), disimbolkan dengan Z, atau kadang-kadang disebut juga vektor inisialisasi (IV), agar diperoleh kondisi awal yang berbeda pada setiap lelaran (lihat Gambar 7.3).



Gambar 7.3 Cipher aliran dengan pembangkit bit-aliran kunci yang bergantung pada kunci U dan umpan Z.

Dengan demikian, bit-bit kunci K dapat dinyatakan sebagai hasil dari fungsi g dengan parameter kunci U dan masukan umpan Z:

$$K = g_K(Z)$$

sehingga proses enkripsi dan dekripsi didefinisikan

$$C = P \oplus K = P \oplus g_K(Z)$$

$$P = C \oplus K = C \oplus g_K(Z)$$

Nilai *Z* yang berbeda-beda pada setiap lelaran menghasilkan bit-bit kunci yang berbeda pula.

Menggunakan pasangan *Z* dan *U* yang sama dua kali dapat menyebabkan bit-bit kunci (*keystream*) yang sama pada setiap kali. Penggunaan *keystream* yang sama dua kali memudahkan jenis penyerangan *ciphertext attack* (aka diijelaskan kemudian).

Karena bit-bit kunci hanya bergantung pada *Z* dan *U*, maka bit-bit kunci ini tidak terpengaruh oleh kesalahan transmisi di dalam cipherteks. Jadi, kesalahan 1-bit pada transmisi cipherteks hanya menghasilkan kesalahan 1-bit pada plainteks hasil dekripsi

Serangan yang dapat dilakukan oleh kriptanalis terhadap cipher aliran adalah:

1. *Known-plaintext attack*

   Misalkan kriptanalis memiliki potongan plainteks (P) dan cipherteks (C) yang berkoresponden, maka ia dapat menemukan bagian bit-aliran-kunci (K) yang berkoresponden dengan meng-XOR-kan bit-bit plainteks dan cipherteks:

$$\begin{aligned} P \oplus C &= P \oplus (P \oplus K) \\ &= (P \oplus P) \oplus K \\ &= 0 \oplus K \\ &= K \end{aligned}$$

Contoh 7.3 :

P	01100101		(karakter 'e')
K	00110101	$\oplus$	(karakter '5')
C	01010000		(karakter 'P')
P	01100101	$\oplus$	(karakter 'e')
K	00110101		(karakter '5')

2. *Ciphertext-only attack*

   Serangan ini terjadi jika keystream yang sama digunakan dua kali terhadap potongan plainteks yang berbeda. Serangan semacam ini disebut juga keystream reuse attack. Misalkan kriptanalis memiliki dua potongan cipherteks berbeda (C1 dan C2) yang dienkripsi dengan bit-bit kunci yang sama. Ia meng-XOR-kan kedua cipherteks tersebut dan memperoleh dua buah plainteks yang ter-XOR satu sama lain:

$$\begin{aligned} C_1 \oplus C_2 &= (P_1 \oplus K) \oplus (P_2 \oplus K) \\ &= (P_1 \oplus P_2) \oplus (K \oplus K) \\ &= (P_1 \oplus P_2) \oplus 0 \\ &= (P_1 \oplus P_2) \end{aligned}$$

   Jika salah satu dari P1 atau P2 diketahui atau dapat diterka, maka XOR-kan salah satu plainteks tersebut dengan cipherteksnya untuk memperoleh bit-bit kunci K yang berkoresponden:

$$P_1 \oplus C_1 = P_1 \oplus (P_1 \oplus K) = K$$

   Selanjutnya P2 dapat diungkap dengan kunci K ini.

   Jika P1 atau P2 tidak diketahui, dua buah plainteks yang ter-XOR satu sama lain ini dapat diketahui dengan menggunakan nilai statistik dari pesan. Misalnya dalam

teks Bahasa Inggris, dua buah spasi ter-XOR, atau satu spasi dengan huruf 'e' yang paling sering muncul, dsb. Kriptanalis cukup cerdas untuk mendeduksi kedua plainteks tersebut.

Contoh 9.4:

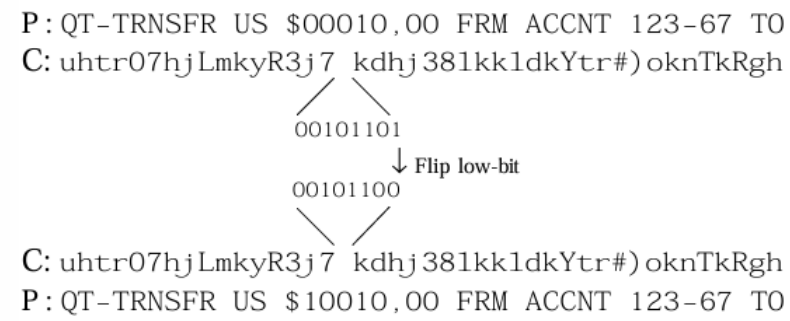
P_1	01100101		(karakter 'e')
K	00110101	$\oplus$	(karakter '5')
C_1	01010000		(karakter 'P')
P_2	01000010		(karakter 'B')
K	00110101	$\oplus$	(karakter '5')
C_2	01110111		(karakter 'w')

$P_1 \oplus P_2 = 01100101 \oplus 01000010 = 00100111$
$C_1 \oplus C_2 = 01010000 \oplus 01110111 = 00100111$

Perhatikan bahwa P1 $\oplus$ P2 = C1 $\oplus$ C2
Jika P1 atau P2 telah diketahui, maka XOR-kan plainteks tersebut dengan cipherteks yang berkoresponden untuk memperoleh K (seperti pada Contoh 7.3).
Pesan moral dari dua serangan di atas adalah: pengguna cipher aliran harus mempunyai bit-bit kunci yang tidak dapat diprediksi sehingga mengetahui sebagian dari bit-bit kunci kunci tidak memungkinkan kriptanalis dapat mendeduksi bagian sisanya.

3. *Flip-bit attack*
   Serangan ini tidak bertujuan menemukan kunci atau mengungkap plainteks dari cipherteks, tetapi mengubah bit cipherteks tertentu sehingga hasil dekripsinya berubah. Pengubahan dilakukan dengan membalikkan (flip) bit tertentu (0 menjadi 1, atau 1 menjadi 0).

Contoh 7.5:



Pengubahan 1 bit U dari cipherteks sehingga menjadi T. Hasil dekripsi: $10,00 menjadi $ 10010,00

Pengubah pesan tidak perlu mengetahui kunci, ia hanya perlu mengetahui posisi pesan yang diminati saja.

Serangan semacam ini memanfaatkan karakteristik cipher aliran yang sudah disebutkan di atas, bahwa kesalahan 1-bit pada cipherteks hanya menghasilkan kesalahan 1-bit pada plainteks hasil dekripsi.

Cipher aliran cocok untuk mengenkripsikan aliran data yang terus menerus melalui saluran komunikasi, misalnya:

1. Mengenkripsikan data pada saluran yang menghubungkan antara dua buah komputer. 2.
2. Mengenkripsikan suara pada jaringan telepon mobile GSM.

Alasannya, jika bit cipherteks yang diterima mengandung kesalahan, maka hal ini hanya menghasilkan satu bit kesalahan pada waktu dekripsi, karena tiap bit plainteks ditentukan hanya oleh satu bit cipherteks. Kondisi ini tidak benar untuk cipher blok karena jika satu bit cipherteks yang diterima mengandung kesalahan, maka kesalahan ini akan merambat pada seluruh blok bit plainteks hasil dekripsi (error propagation).

8.
Tipe dan Mode **Algoritma Simetri (Bagian 2)**

8.1. Cipher Blok (Block Cipher)

Pada cipher blok, rangkaian bit-bit plainteks dibagi menjadi blok-blok bit dengan panjang sama, biasanya 64 bit (tapi adakalanya lebih).

Enkripsi dilakukan terhadap blok bit plainteks menggunakan bit-bit kunci (yang ukurannya sama dengan ukuran blok plainteks). Algoritma enkripsi menghasilkan blok cipherteks yang berukuran sama dengan blok plainteks.

Dekripsi dilakukan dengan cara yang serupa seperti enkripsi. Misalkan blok plainteks (P) yang berukuran m bit dinyatakan sebagai vektor

$$P = (p_1, p_2, \ldots, p_m)$$

yang dalam hal ini p_i adalah 0 atau 1 untuk i = 1, 2, …, m, dan blok cipherteks (C) adalah

$$C = (c_1, c_2, \ldots, c_m)$$

yang dalam hal ini c_i adalah 0 atau 1 untuk i = 1, 2, …, m.
Bila plainteks dibagi menjadi n buah blok, barisan blok-blok plainteks dinyatakan sebagai

$$(P_1, P_2, \ldots, P_n)$$

Untuk setiap blok plainteks Pi, bit-bit penyusunnya dapat dinyatakan sebagai vektor

$$P_i = (p_{i1}, p_{i2}, \ldots, p_{im})$$

Enkripsi dan dekripsi dengan kunci K dinyatakan berturut turut dengan persamaan

$$E_K(P) = C$$

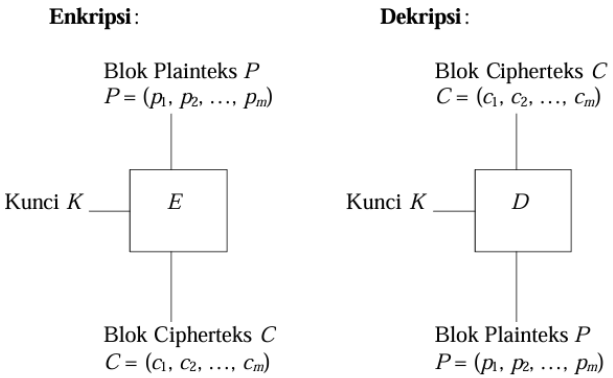
untuk enkripsi, dan

$$D_K(C) = P$$

Fungsi E haruslah fungsi yang berkoresponden satu-ke-satu, sehingga

$$E^1 = D$$

Skema enkripsi dan dekripsi dengan cipher blok digambarkan pada Gambar 8.1. Fungsi E dan D dispesifikasikan oleh kriptografer.



Gambar 8.1 Skema enkripsi dan dekripsi pada cipher blok

8.2. Teknik Kriptografi Klasik yang Digunakan pada Cipher Blok

Algoritma blok cipher menggabungkan beberapa teknik kriptografi klasik dalam proses enkripsi. Dengan kata lain, cipher blok dapat diacu sebagai super-enkripsi. Teknik kriptografi klasik yang digunakan adalah:

1. Substitusi.
   Teknik ini mengganti satu atau sekumpulan bit pada blok plainteks tanpa mengubah urutannya. Secara matematis, teknik substitusi ini ditulis sebagai

   ci = E(pi), i = 1, 2, … (urutan bit)

   yang dalam hal ini ci adalah bit cipherteks, pi adalah bit plainteks, dan f adalah fungsi substitusi. Dalam praktek, E dinyatakan sebagai fungsi matematis atau dapat merupakan tabel susbtitusi (S-box).
2. Transposisi atau permutasi
   Teknik ini memindahkan posisi bit pada blok plainteks berdasarkan aturan tertentu. Secara matematis, teknik transposisi ini ditulis sebagai
   C = PM
   yang dalam hal ini C adalah blok cipherteks, P adalah blok plainteks, dan M adalah fungsi transposisi. Dalam praktek, M dinyatakan sebagai tabel atau matriks permutasi.

Selain kedua teknik di atas, cipher blok juga menggunakan dua teknik tambahan sebagai berikut:

3. Ekspansi
   Teknik ini memperbanyak jumlah bit pada blok plainteks berdasarkan aturan tertentu, misalnya dari 32 bit menjadi 48 bit. Dalam praktek, aturan eskpansi dinyatakan dengan tabel.
4. Kompresi

Teknik ini kebalikan dari ekspansi, di mana jumlah bit pada blok plainteks diciutkan berdasarkan aturan tertentu. Dalam praktek, aturan kompresi dinyatakan dengan tabel.

8.3. Prinsip Penyandian Shannon

Pada tahun 1949, Shannon mengemukakan dua prinsip (properties) penyandian (encoding) data. Kedua prinsip ini dipakai dalam perancangan cipher blok yang kuat. Kedua prinsip Shannon tersebut adalah:

1. Confusion

   Prinsip ini menyembunyikan hubungan apapun yang ada antara plainteks, cipherteks, dan kunci. Sebagai contoh, pada cipher substitusi seperti caesar cipher, hubungan antara cipherteks dan plainteks mudah diketahui, karena satu huruf yang sama pada plainteks diganti dengan satu huruf yang sama pada cipherteksnya.

   Prinsip confusion akan membuat kriptanalis frustasi untuk mencari pola-pola statistik yang muncul pada cipherteks. Confusion yang bagus membuat hubungan statistik antara plainteks, cipherteks, dan kunci menjadi sangat rumit.

2. Diffusion

   Prinsip ini menyebarkan pengaruh satu bit plainteks atau kunci ke sebanyak mungkin cipherteks. Sebagai contoh, pengubahan kecil pada plainteks sebanyak satu atau dua bit menghasilkan perubahan pada cipherteks yang tidak dapat diprediksi.

   Prinsip diffusion juga menyembunyikan hubungan statistik antara plainteks, cipherteks, dan kunci dan membuat kriptanalisis menjadi sulit.

Untuk mendapatkan keamanan yang bagus, prinsip confusion dan diffusion diulang berkali-kali pada sebuah blok tunggal dengan kombinasi yang berbeda-beda.

8.4. Mode Operasi Cipher Blok

Plainteks dibagi menjadi beberapa blok dengan panjang tetap. Beberapa mode operasi dapat diterapkan untuk melakukan enkripsi terhadap keseluruhan blok plainteks. Empat mode operasi yang lazim diterapkan pada sistem blok cipher adalah:

1. Electronic Code Book (ECB)
2. Cipher Block Chaining (CBC)
3. Cipher Feedback (CFB)
4. Output Feedback (OFB)

8.5. Electronic Code Book (ECB)

Pada mode ini, setiap blok plainteks Pi dienkripsi secara individual dan independen meknajdi blok cipherteks Ci . Secara matematis, enkripsi dengan mode ECB dinyatakan sebagai

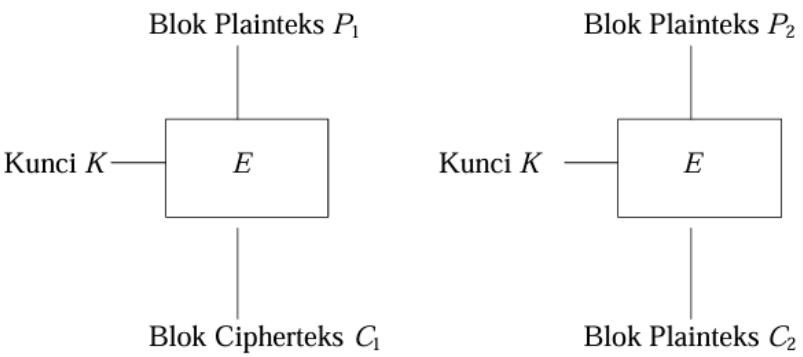
$$C_i = E_K(P_i)$$

dan dekripsi sebagai

$$P_i = D_K(C_i)$$

yang dalam hal ini, Pi dan Ci masing-masing blok plainteks dan cipherteks ke-i.

Gambar 8.2 memperlihatkan enkripsi dua buah blok plainteks, P1 dan P2 dengan mode ECB, yang dalam hal ini E menyatakan fungsi enkripsi yang melakukan enkripsi terhadap blok plainteks dengan menggunakan kunci K.



Gambar 8.2 Skema enkripsi dan dekripsi dengan mode ECB

Contoh 8.1: Misalkan plainteks (dalam biner) adalah

10100010001110101001

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.

Misalkan fungsi enkripsi E yang sederhana (tetapi lemah) adalah dengan meng-XOR-kan blok plainteks Pi dengan K, kemudian geser secara wrapping bit-bit dari Pi $\oplus$ K satu posisi ke kiri. Proses enkripsi untuk setiap blok digambarkan sebagai berikut:

	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

1010001000111010101001 (A23A9 dalam notasi HEX)

Adalah

00100011000100100100 (23124 dalam notasi HEX)

Catatlah bahwa blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama (atau identik). Pada contoh 1 di atas, blok 1010 muncul dua kali dan selalu dienkripsi menjadi 0010.

Contoh yang lebih nyata misalkan pesan

KUTU BUKU DI LEMARIKU

dibagi menjadi blok-blok yang terdiri dua huruf (dengan menghilangkan semua spasi) menjadi

KU TU BU KU DI LE MA RI KU

maka blok yang menyatakan "KU" akan dienkripsi menjadi blok cipherteks (dua huruf) yang sama.

Kata "code book" di dalam ECB muncul dari fakta bahwa karena blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama, maka secara teoritis dimungkinkan membuat buku kode plainteks dan cipherteks yang berkoresponden.

Namun, semakin besar ukuran blok, semakin besar pula ukuran buku kodenya. Misalkan jika blok berukuran 64 bit, maka buku kode terdiri dari 2^{64} – 1 buah kode (entry), yang berarti terlalu besar untuk disimpan. Lagipula, setiap kunci mempunyai buku kode yang berbeda.

Padding

Ada kemungkinan panjang plainteks tidak habis dibagi dengan panjang ukuran blok yang ditetapkan (misalnya 64 bit atau lainnya). Hal ini mengakibatkan blok terakhir berukuran lebih pendek daripada blok-blok lainnya.

Satu cara untuk mengatasi hal ini adalah dengan padding, yaitu menambahkan blok terakhir dengan pola bit yang teratur agar panjangnya sama dengan ukuran blok yang ditetapkan. Misalnya ditambahkan bit 0 semua, atau bit 1 semua, atau bit 0 dan bit 1 berselang-seling.

Misalkan ukuran blok adalah 64 bit (8 byte) dan blok terakhir terdiri dari 24 bit (3 byte). Tambahkan blok terakhir dengan 40 bit (5 byte) agar menjadi 64 bit, misalnya dengan menambahkan 4 buah byte 0 dan satu buah byte angka 5. Setelah dekripsi, hapus 5 byte terakhir dari blok dekripsi terakhir.

Keuntungan Mode ECB

1. Karena tiap blok plainteks dienkripsi secara independen, maka kita tidak perlu mengenkripsi file secara linear. Kita dapat mengenkripsi 5 blok pertama, kemudian blok-blok di akhir, dan kembali ke blok-blok di tengah dan seterusnya.
   Mode ECB cocok untuk mengenkripsi arsip (file) yang diakses secara acak, misalnya arsip-arsip basisdata. Jika basisdata dienkripsi dengan mode ECB, maka sembarang record dapat dienkripsi atau didekripsi secara independen

dari record lainnya (dengan asumsi setiap record terdiri dari sejumlah blok diskrit yang sama banyaknya).
Jika mode ECB dikerjakan dengan prosesor paralel (multiple processor), maka setiap prosesor dapat melakukan enkripsi atau dekripsi blok plainteks yang berbeda-beda.

2. Jika satu atau lebih bit pada blok cipherteks mengalami kesalahan, maka kesalahan ini hanya mempengaruhi cipherteks yang bersangkutan pada waktu dekripsi. Blok-blok cipherteks lainnya bila didekripsi tidak terpengaruh oleh kesalahan bit cipherteks tersebut.

Kelemahan ECB

1. Karena bagian plainteks sering berulang (sehingga terdapat blok-blok plainteks yang sama), maka hasil enkripsinya menghasilkan blok cipherteks yang sama (lihat Contoh 1).
Bagian plainteks yang sering berulang misalnya kata-kata seperti (dalam Bahasa Indonesia) dan, yang, ini, itu, dan sebagainya.
Di dalam e-mail, pesan sering mengandung bagian yang redundan seperti string 0 atau spasi yang panjang, yang bila dienkripsi maka akan menghasilkan pola-pola cipherteks yang mudah dipecahkan dengan serangan yang berbasis statistik (menggunakan frekuensi kemunculan blok cipherteks). Selain itu, e-mail mempunyai struktur yang teratur yang menimbulkan pola-pola yang khas dalam cipherteksnya.
Misalnya kriptanalis mempelajari bahwa blok plainteks 5EB82F (dalam notasi HEX) dienkripsi menjadi blok AC209D, maka setiap kali ia menemukan cipherteks

AC209D, ia dapat langsung mendekripsinya menjadi 5EB82F.

Satu cara untuk mengurangi kelemahan ini adalah menggunakan ukuran blok yang besar, misalnya 64 bit, sebab ukuran blok yang besar dapat menghilangkan kemungkinan menghasilkan blok-blok yang identik.

2. Pihak lawan dapat memanipulasi cipherteks untuk "membodohi" atau mengelabui penerima pesan.

Contoh 8.2.

Misalkan seseorang mengirim pesan

Uang ditransfer lima satu juta rupiah

Andaikan bahwa kriptanalis mengetahui bahwa blok plainteks terdiri dari dua huruf (spasi diabaikan sehingga menjadi 16 blok plainteks) dan blok-blok cipherteksnya adalah

$$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}$$

Misalkan kriptanalis berhasil mendekripsi keseluruhan blok cipherteks menjadi plainteks semula, sehingga ia dapat mendekripsi C1 menjadi Ua, C2 menjadi ng, C3 menjadi di dan seterusnya. Kriptanalis memanipulasi cipherteks dengan membuang blok cipheteks ke-8 dan 9 sehingga menjadi

$$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}$$

Penerima pesan mendekripsi cipherteks yang sudah dimanipulasi dengan kunci yang benar menjadi

Uang ditransfer satu juta rupiah

Karena dekripsi menghasilkan pesan yang bermakna, maka penerima menyimpulkan bahwa uang yang dikirim kepadanya sebesar satu juta rupiah.

Kedua kelemahan di atas dapat diatasi dengan mengatur enkripsi tiap blok individual bergantung pada semua blok blok sebelumnya. Dengan cara ini, blok plainteks yang identik akan menghasilkan blok cipherteks yang berbeda, dan manipulasi cipherteks mungkin menghasilkan pesan hasil dekripsi yang tidak mempunyai makna. Prinsip inilah yang mendasari mode operasi cipher blok yang kedua, yaitu Cipher Block Chaining.

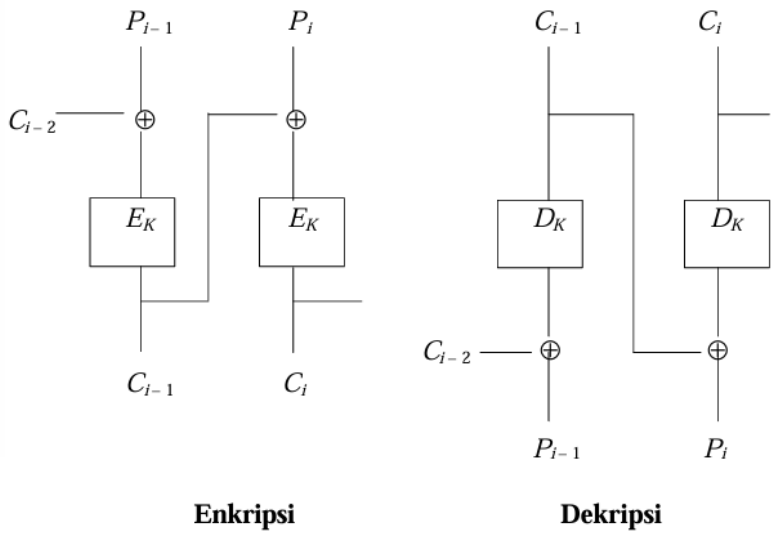
8.6. Cipher Block Chaining (CBC)

Mode ini menerapkan mekanisme umpan-balik (feedback) pada sebuah blok, yang dalam hal ini hasil enkripsi blok sebelumnya di-umpan-balikkan ke dalam enkripsi blok yang current.

Caranya, blok plainteks yang current di-XOR-kan terlebih dahulu dengan blok cipherteks hasil enkripsi sebelumnya, selanjutnya hasil peng-XOR-an ini masuk ke dalam fungsi enkripsi.

Dengan mode CBC, setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya tetapi juga pada seluruh blok plainteks sebelumnya.

Dekripsi dilakukan dengan memasukkan blok cipherteks yang current ke fungsi dekripsi, kemudian meng-XOR-kan hasilnya dengan blok cipherteks sebelumnya. Dalam hal ini, blok cipherteks sebelumnya berfungsi sebagai umpan-maju (feedforward) pada akhir proses dekripsi. Gambar 8.3 memperlihatkan skema mode operasi CBC.



Gambar 8.3 Skema enkripsi dan dekripsi dengan mode CBC

Secara matematis, enkripsi dengan mode CBC dinyatakan sebagai

$$C_i = E_K(P_i \oplus C_{i-1})$$

dan dekripsi sebagai

$$P_i = D_K(C_i) \oplus C_{i-1}$$

Yang dalam hal ini, C_0 = *IV (initialization vector)*. *IV* dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh program.

Jadi, untuk menghasilkan blok cipherteks pertama (C_1) , *IV* digunakan untuk menggantikan blok cipherteks sebelumnya, C_0.

Sebaliknya pada dekripsi, blok plainteks diperoleh dengan cara meng-*XOR*-kan *IV* dengan hasil dekripsi terhadap blok cipherteks pertama.

Perhatikan bahwa enkripsi terhadap blok *i* adalah fungsi dari semua plainteks dari blok 0 sampai blok *i* – 1, sehingga blok

plianteks yang sama menghasilkan blok cipherteks yang berbeda hanya jika blok-blok plainteksnya sebelumnya berbeda.

Jika blok-blok plainteks sebelumnya ada yang sama, maka ada kemungkinan cipherteksnya sama. Untuk mencegah hal ini, maka digunakan *IV* yang merupakan data acak sebagai blok pertama. *IV* tidak mempunyai makna, ia hanya diguanakan untuk membuat tiap blok cipherteks menjadi unik.

Contoh 8.3.

Tinjau kembali plainteks dari Contoh 8.1:

10100010001110101001

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)
Misalkan fungsi enkripsi E yang sederhana (tetapi lemah) adalah dengan meng-XOR-kan blok plainteks Pi dengan K, kemudian geser secara wrapping bit-bit dari Pi $\oplus$ K satu posisi ke kiri.

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, $C_2 = 1111$ (atau F dalam HEX)

Demikian seterusnya, sehingga plainteks dan cipherteks hasilnya adalah:

Pesan (plainteks):	A23A9
Cipherteks (mode ECB):	23124
Cipherteks (mode CBC):	27FBF

Terlihat bahwa dengan menggunakan mode CBC, blok plainteks yang sama (A dalam HEX) dienkripsikan menjadi dua blok cipherteks yang berbeda (masing-masing 2 dan B). Bandingkan dengan mode EBC yang menghasilkan blok cipherteks yang sama (2 dalam HEX) untuk dua buah blok yang sama (A).

Dengan kata lain, pada mode CBC, tidak ada korelasi antara posisi blok plainteks yang sama dengan posisi blok cipherteksnya.

Keuntungan Mode CBB

- Karena blok-blok plainteks yang sama tidak menghasilkan blok-blok cipherteks yang sama, maka kriptanalisis menjadi lebih sulit. Inilah alasan utama penggunaan mode CBC digunakan.

Kelemahan Mode CBC

- Karena blok cipherteks yang dihasilkan selama proses enkripsi bergantung pada blok-blok cipherteks sebelumnya, maka kesalahan satu bit pada sebuah blok plainteks akan merambat pada blok cipherteks yang berkoresponden dan semua blok cipherteks berikutnya.
- Tetapi, hal ini berkebalikan pada proses dekripsi. Kesalahan satu bit pada blok cipherteks hanya mempengaruhi blok plainteks yang berkoresponden dan satu bit pada blok plainteks berikutnya (pada posisi bit yang berkoresponden pula).
- Kesalahan bit cipherteks biasanya terjadi karena adanya gangguan (noise) saluran komunikasi data selama transmisi atau malfunction pada media penyimpanan.

Persoalan Keamanan yang Muncul pada Mode CBC

1. Karena blok cipherteks mempengaruhi blok-blok berikutnya, pihak lawan dapat menambahkan blok cipherteks tambahan pada akhir pesan terenkripsi tanpa terdeteksi. Ini akan menghasilkan blok plainteks tambahan pada waktu dekripsi.

   Pesan moral untuk masalah ini, pengirim pesan seharusnya menstrukturkan plainteksnya sehingga ia mengetahui di mana ujung pesan dan dapat mendeteksi adanya blok tambahan.
2. Pihak lawan dapat mengubah cipherteks, misalnya mengubah sebuah bit pada suatu blok cipherteks. Tetapi

hal ini hanya mempengaruhi blok plainteks hasil dekripsinya dan satu bit keslahan pada posisi plainteks berikutnya.

9.
Data Encryption Standard **(DES)**

9.1. Sejarah DES

Data Encryption Standard DES) dikembangkan di IBM di bawah pimpinan W.L. Tuchman pada tahun 1972. Algoritma ini merupakan pengembangan dari algoritma *Lucifer*, yang awalnya dirancang oleh Horst Feistel. DES dirancang untuk menjadi algoritma enkripsi simetris yang kuat untuk melindungi informasi sensitif melalui enkripsi berbasis kunci yang menghasilkan ciphertext dari plaintext menggunakan metode yang kompleks.

National Bureau of Standards (NBS), yang sekarang dikenal sebagai *National Institute of Standards and Technology* (NIST), mengadopsi DES sebagai standar enkripsi resmi untuk data sensitif pada tahun 1977. Sebelum standardisasi ini, algoritma DES dievaluasi oleh *National Security Agency* (NSA) Amerika Serikat untuk memastikan kekuatannya terhadap serangan kriptografi. Meskipun telah menjadi standar enkripsi yang dominan selama beberapa dekade, penggunaan DES telah menurun karena panjang kunci 56-bit-nya dianggap rentan terhadap serangan *brute force* seiring dengan peningkatan daya komputasi.

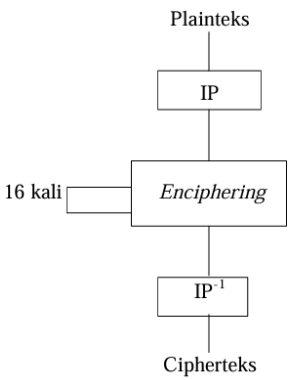
DES memainkan peran utama dalam pengembangan kriptografi modern dan menjadi dasar untuk algoritma enkripsi yang lebih kuat, seperti Triple DES dan *Advanced Encryption Standard* (AES), yang menggantikan DES sebagai standar enkripsi modern untuk keamanan data yang lebih tinggi.

9.2. DES

DES termasuk ke dalam sistem kriptografi simetri dan tergolong jenis cipher blok. DES beroperasi pada ukuran blok 64 bit. DES mengenkripsikan 64 bit plainteks menjadi 64 bit cipherteks dengan menggunakan 56 bit kunci internal (internal key) atau upa-kunci (subkey). Kunci internal dibangkitkan dari kunci eksternal (external key) yang panjangnya 64 bit.

Skema global dari algoritma DES adalah sebagai berikut (lihat Gambar 9.1):

1. Blok plainteks dipermutasi dengan matriks permutasi awal (initial permutation atau IP).
2. Hasil permutasi awal kemudian di-enciphering- sebanyak 16 kali (16 putaran). Setiap putaran menggunakan kunci internal yang berbeda.
3. Hasil enciphering kemudian dipermutasi dengan matriks permutasi balikan (invers initial permutation atau IP-1) menjadi blok cipherteks.



Gambar 9.1 Skema Global Algoritma DES

Di dalam proses enciphering, blok plainteks terbagi menjadi dua bagian, kiri (L) dan kanan (R), yang masing-masing

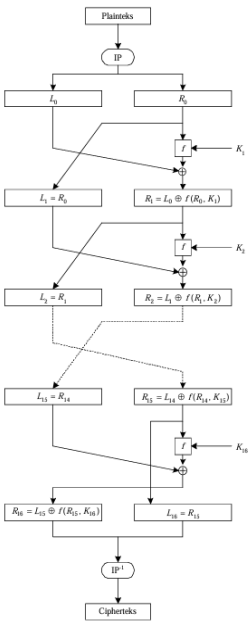
panjangnya 32 bit. Kedua bagian ini masuk ke dalam 16 putaran DES.

Pada setiap putaran i, blok R merupakan masukan untuk fungsi transformasi yang disebut f. Pada fungsi f, blok R dikombinasikan dengan kunci internal Ki. Keluaran dai 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. Ini adalah satu putaran DES.

Secara matematis, satu putaran DES dinyatakan sebagai

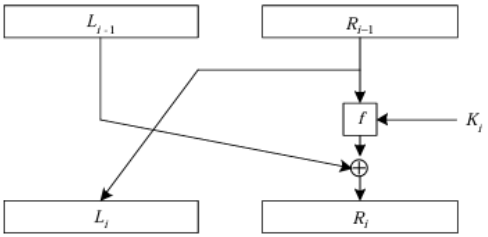
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Gambar 9.2 memperlihatkan skema algoritma DES yang lebih rinci.



Gambar 9.2 Algoritma Enkripsi dengan DS

Satu putaran DES merupakan model jaringan Feistel (lihat Gambar 9.3).



Gambar 9.3 Jaringan Feistel untuk satu putaran DES

Perlu dicatat dari Gambar 12.2 bahwa jika (L16, R16) merupakan keluaran dari putaran ke-16, maka (R16, L16) merupakan pra-cipherteks (pre-ciphertext) dari enciphering ini. Cipherteks yang sebenarnya diperoleh dengan melakukan permutasi awal balikan, IP-1, terhadap blok pra-cipherteks.

9.3. Permutasi Awal

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

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

Cara membaca tabel/matriks: dua entry ujung kiri atas (58 dan 50) berarti: "pindahkan bit ke-58 ke posisi bit 1"

"pindahkan bit ke-50 ke posisi bit 2", dst

9.4. Pembangkitan Kunci Internal

Karena ada 16 putaran, maka dibutuhkan kunci internal sebanyak 16 buah, yaitu K_1, K_2, …, K_{16}. Kunci-kunci internal ini

dapat dibangkitkan sebelum proses enkripsi atau bersamaan dengan proses enkripsi.

Kunci internal dibangkitkan dari kunci eksternal yang diberikan oleh pengguna. Kunci eksternal panjangnya 64 bit atau 8 karakter.

Misalkan kunci eksternal yang tersusun dari 64 bit adalah *K*. Kunci eksternal ini menjadi masukan untuk permutasi dengan menggunakan matriks permutasi kompresi PC-1 sebagai berikut:

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

Dalam permutasi ini, tiap bit kedelapan (parity bit) dari delapan byte kunci diabaikan. Hasil permutasinya adalah sepanjang 56 bit, sehingga dapat dikatakan panjang kunci DES adalah 56 bit.

Selanjutnya, 56 bit ini dibagi menjadi 2 bagian, kiri dan kanan, yang masing-masing panjangnya 28 bit, yang masing masing disimpan di dalam C_0 dan D_0:

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

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18
10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36

D_0: berisi bit-bit dari K pada posisi

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22
14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

Selanjutnya, kedua bagian digeser ke kiri (left shift) sepanjang satu atau dua bit bergantung pada tiap putaran. Operasi pergeseran bersifat wrapping atau round-shift. Jumlah pergeseran pada setiap putaran ditunjukkan pada Tabel 9.1 sbb:

Tabel 9.1 Jumlah pergeseran bit pada setiap putaran

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

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

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

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

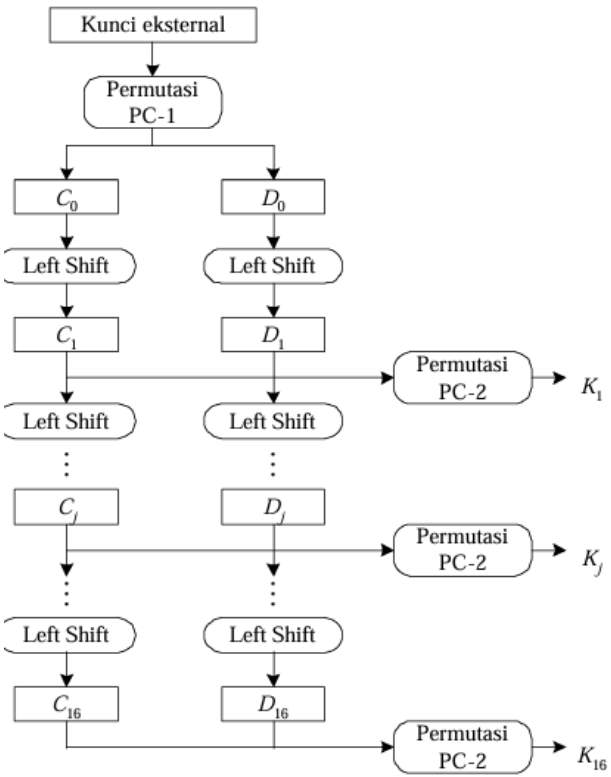
14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10
23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2

dengan bit-bit D_i pada posisi:

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48
44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

Jadi, setiap kunci internal K_i mempunyai panjang 48 bit.
Proses pembangkitan kunci-kunci internal ditunjukkan pada Gambar 9.4.

Bila jumlah pergeseran bit-bit pada Tabel 1 dijumlahkan semuanya, maka jumlah seluruhnya sama dengan 28, yang sama dengan jumlah bit pada C_i dan D_i. Karena itu, setelah putaran ke-16 akan didapatkan kembali $C_{16} = C_0$ dan $D_{16} = D_0$.



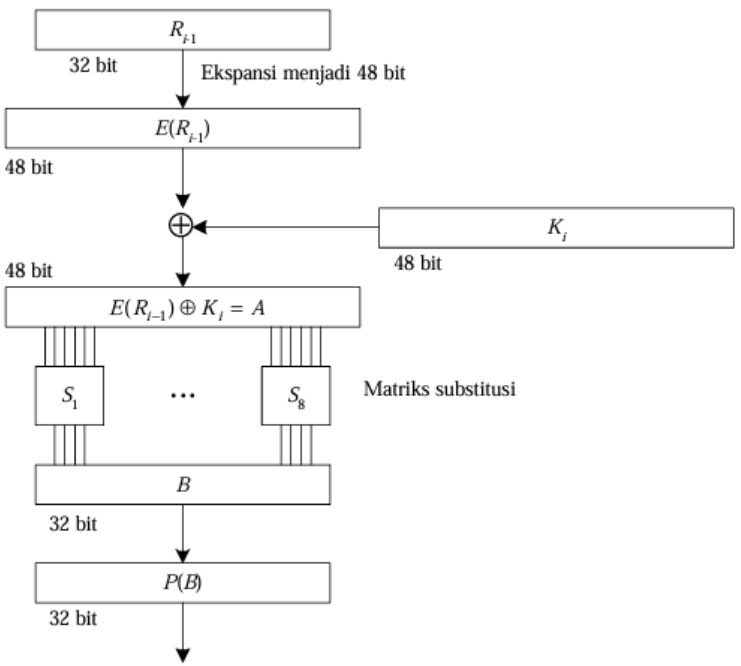
Gambar 9.4 Proses pembangkitan kunci-kunci internal DES

9.5. Enciphering

Proses enciphering terhadap blok plainteks dilakukan setelah permutasi awal (lihat Gambar 9.1). Setiap blok plainteks mengalami 16 kali putaran enciphering (lihat Gambar 2). Setiap putaran enciphering merupakan jaringan Feistel yang secara matematis dinyatakan sebagai

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

Diagram komputasi fungsi f diperlihatkan pada Gambar 9.5.



Gambar 9.5 Rincian komputasi fungsi *f*

E adalah fungsi ekspansi yang memperluas blok R_{i-1} yang panjangnya 32-bit menjadi blok 48 bit. Fungsi ekspansi direalisasikan dengan matriks permutasi ekspansi sbb:

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

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

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

Vektor A dikelompokkan menjadi 8 kelompok, masing masing 6 bit, dan menjadi masukan bagi proses substitusi. Proses substitusi dilakukan dengan menggunakan delapan buah kotak-S (*S-box*), S_1 sampai S_8. Setiap kotak-S menerima masukan 6 bit dan menghasilkan keluaran 4 bit. Kelompok 6 bit pertama menggunakan S_1, kelompok 6-bit kedua menggunakan S_2, dan seterusnya. (*cara pensubstitusian dengan kotak-S sudah dijelaskan pada materi "Tipe dan Mode Algoritma Simetri (Bagian 3)"*)

Kedelapan kotak-S tersebut adalah:

S_1:

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

S_2:

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

S_3:

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

S_4:

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

S_5:

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	16
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6:

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

S_7:

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

S_8:

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

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

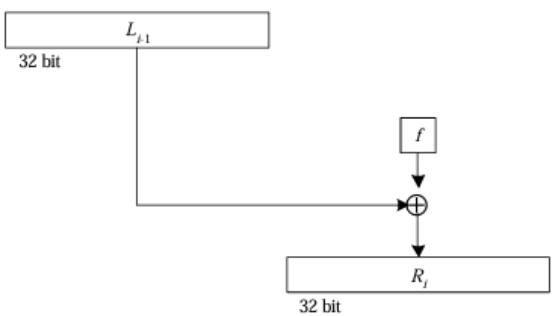
16	7	20	21	29	12	28	17	1	15	23	26	5	8	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Bit-bit *P*(*B*) merupakan keluaran dari fungsi *f*. Akhirnya, bit-bit *P*(*B*) di-XOR-kan dengan L_{i-1} untuk mendapatkan R_i (lihat Gambar 6):

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

Jadi, keluaran dari putaran ke-*i* adalah

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



Gambar 9.6 Skema perolehan R_i

9.6. Permutasi Terakhir (*Inverse Initial Permutation*)

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

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

9.7. Dekripsi

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

Untuk tiap putaran 16, 15, …, 1, keluaran pada setiap putaran deciphering adalah

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

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

Tinjau kembali proses pembangkitan kunci internal pada Gambar 4. Selama deciphering, K_{16} dihasilkan dari (C_{16}, D_{16}) dengan permutasi PC-2. Tentu saja (C_{16}, D_{16}) tidak dapat diperoleh langsung pada permulaan deciphering. Tetapi karena $(C_{16}, D_{16}) = (C_0, D_0)$, maka K_{16} dapat dihasilkan dari $(C0, D0)$ tanpa perlu lagi melakukan pergeseran bit. Catatlah bahwa (C_0, D_0) yang

merupakan bit-bit dari kunci eksternal K yang diberikan pengguna pada waktu dekripsi.

Selanjutnya, K_{15} dihasilkan dari (C_{15}, D_{15}) yang mana (C_{15}, D_{15}) diperoleh dengan menggeser C_{16} (yang sama dengan C_0) dan D_{16} (yang sama dengan C_0) satu bit ke kanan. Sisanya, K_{14} sampai K_1 dihasilkan dari (C_{14}, D_{14}) sampai (C_1, D_1). Catatlah bahwa (C_{i-1} ,D_{i-1}) diperoleh dengan menggeser Ci dan Di dengan cara yang sama seperti pada Tabel 1, tetapi pergeseran kiri (*left shift*) diganti menjadi pergeseran kanan (*right shift*).

9.8. Mode DES

DES dapat dioperasikan dengan mode ECB, CBC, OFB, dan CFB. Namun karena kesederhanaannya, mode ECB lebih sering digunakan pada paket program komersil meskipun sangat rentan terhadap serangan.

Mode CBC lebih kompleks daripada EBC namun memberikan tingkat keamanan yang lebih bagus daripada mode EBC. Mode CBC hanya kadang-kadang saja digunakan.

9.9. Implementasi *Hardware* dan *Software* DES

DES sudah diimplementasikan dalam bentuk perangkat keras. Dalam bentuk perangkat keras, DES diimplementasikan di dalam *chip*. Setiap detik *chip* ini dapat mengenkripsikan 16,8 juta blok (atau 1 gigabit per detik).

Implementasi DES ke dalam perangkat lunak dapat melakukan enkripsi 32.000 blok per detik (pada komputer *mainframe* IBM 3090).

9.10. Keamanan DES

Isu-isu yang menjadi perdebatan kontroversial menyangkut keamanan DES:

1. Panjang kunci
2. Jumlah putaran

3. Kotak-S

Panjang kunci

Panjang kunci eksternal DES hanya 64 bit atau 8 karakter, itupun yang dipakai hanya 56 bit. Pada rancangan awal, panjang kunci yang diusulkan IBM adalah 128 bit, tetapi atas permintaan NSA, panjang kunci diperkecil menjadi 56 bit. Alasan pengurangan tidak diumumkan.

Tetapi, dengan panjang kunci 56 bit akan terdapat 256 atau 72.057.594.037.927.936 kemungkinan kunci. Jika diasumsikan serangan *exhaustive key* search dengan menggunakan prosesor paralel mencoba setengah dari jumlah kemungkinan kunci itu, maka dalam satu detik dapat dikerjakan satu juta serangan. Jadi seluruhnya diperlukan 1142 tahun untuk menemukan kunci yang benar.

Tahun 1998, *Electronic Frontier Foundation* (*EFE*) merancang dan membuat perangkat keras khusus untuk menemukan kunci DES secara *exhaustive search key* dengan biaya $250.000 dan diharapkan dapat menemukan kunci selama 5 hari. Tahun 1999, kombinasi perangkat keras *EFE* dengan kolaborasi internet yang melibatkan lebih dari 100.000 komputer dapat menemukan kunci DES kurang dari 1 hari.

Jumlah putaran

Sebenarnya, delapan putaran sudah cukup untuk membuat cipherteks sebagai fungsi acak dari setiap bit plainteks dan setiap bit cipherteks. Jadi, mengapa harus 16 kali putaran?

Dari penelitian, DES dengan jumlah putaran yang kurang dari 16 ternyata dapat dipecahkan dengan *known-plaintext attack* lebih mangkus daripada dengan *brute force attack*.

Kotak-S

Pengisian kotak-S DES masih menjadi misteri tanpa ada alasan mengapa memilih konstanta-konstanta di dalam kotak itu.

DES mempunyai beberapa kunci lemah (*weak key*). Kunci lemah menyebabkan kunci-kunci internal pada setiap putaran sama ($K_1 = K_2 = \ldots = K_{16}$). Akibatnya, enkripsi dua kali berturut-turut terhadap plainteks menghasilkan kembali plainteks semula.

Kunci lemah terjadi bila bit-bit di dalam C_i dan D_i semuanya 0 atau 1, atau setengah dari kunci seluruh bitnya 1 dan setengah lagi seluruhnya 0. Kunci eksternal (dalam notasi HEX) yang menyebabkan terjadinya kunci lemah adalah (ingat bahwa setiap bit kedelapan adalah bit paritas)

Kunci lemah (dengan bit paritas)	Kunci sebenarnya
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 1F1F 1F1F	0000000 FFFFFFF
E0E0 E0E0 F1F1 F11F	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Selain kunci lemah, DES juga mempunyai sejumlah pasangan kunci setengah-lemah (*semiweak key*). Pasangan kunci setengah- lemah mengenkripsikan plainteks menjadi cipherteks yang sama. Sehingga, satu kunci dalam pasangan itu dapat mendekripsi pesan yang dienkripsi oleh kunci yang lain di dalam pasangan itu.

Kunci setengah-lemah terjadi bila:

1. *Register* C dan D berisi bit-bit dengan pola 0101…0101 atau 1010…1010
2. Register yang lain (C atau D) berisi bit-bit dengan pola 0000…0000, 1111…1111, 0101…0101, atau 1010…1010

Ada 6 pasang kunci setengah lemah (dalam notasi HEX):

a. 01FE 01FE 01FE 01FE dan FE01 FE01 FE01 FE01
b. 1FE0 1FE0 0EF1 0EF1 dan E01F E01F F10E F10E
c. 01E0 01E0 01F1 01F1 dan E001 E001 F101 F101
d. 1FFE 1FFE 0EFE 0EFE dan FE1F FE1F FE0E FE0E
e. 011F 011F 010E 010E dan 1F01 1F01 0E01 0E01
f. E0FE E0FE F1FE F1FE dan FEE0 FEE0 FEF1 FEF1

10.
Advanced Encryption Standard (AES)

10.1. Sejarah AES

DES (Data Encryption Standard) mungkin akan berakhir masa penggunaannya sebagai standard enkripsi kriptografi simetri. DES dianggap sudah tidak aman lagi karena dengan perangkat keras khusus kuncinya bisa ditemukan dalam beberapa hari (baca materi kuliah DES). National Institute of Standards and Technology (NIST), sebagai agensi Departemen Perdagangan AS mengusulkan kepada Pemerintah Federal AS untuk sebuah standard kriptografi kriptografi yang baru.

Untuk menghindari kontoversi mengenai standard yang baru tersebut, sebagaimana pada pembuatan DES (NSA sering dicurigai mempunyai "pintu belakang" untuk mengungkap cipherteks yang dihasilkan oleh DES tanpa mengetahui kunci), maka NIST mengadakan sayembara terbuka untuk membuat standard algoritma kriptografi yang baru sebagai pengganti DES. Standard tersebut kelak diberi nama Advanced Encryption Standard (AES).

Persyaratan yang diajukan oleh NIST tentang algoritma yang baru tersebut adalah:

1. Algoritma yang ditawarkan termasuk ke dalam kelompok algoritma kriptografi simetri berbasis cipher blok.
2. Seluruh rancangan algoritma harus publik (tidak dirahasiakan)

3. Panjang kunci fleksibel: 128, 192, dan 256 bit.
4. Ukuran blok yang dienkripsi adalah 128 bit.
5. Algoritma dapat diimplementasikan baik sebagai software maupun hardware.

NIST menerima 15 proposal algoritma yang masuk. Konferensi umum pun diselenggarakan untuk menilai keamanan algoritma yang diusulkan. Pada bulan Agustus 1998, NIST memilih 5 finalis yang didasarkan pada aspek keamanan algoritma, kemangkusan (efficiency), fleksibilitas, dan kebutughan memori (penting untuk embedded system). Finalis tersebut adalah:

1. Rijndael (dari Vincent Rijmen dan Joan Daemen – Belgia, 86 suara)
2. Serpent (dari Ross Anderson, Eli Biham, dan Lars Knudsen Inggris, Israel, dan Norwegia, 59 suara).
3. Twofish (dari tim yang diketuai oleh Bruce Schneier – USA, 31 suara)
4. RC6 (dari Laboratorium RSA – USA, 23 suara)
6. MARS (dari IBM, 13 suara)

Pada bulan Oktober 2000, NIST mengumumkan untuk memilih Rijndael (dibaca: Rhine-doll), dan pada bulan November 2001, Rijndael ditetapkan sebagai AES, dan diharapkan Rijndael menjadi standard kriptografi yang dominan paling sedikit selama 10 tahun.

10.2. Panjang Kunci dan Ukuran Blok Rijndael

Rijndael mendukung panjang kunci 128 bit sampai 256 bit dengan step 32 bit. Panjang kunci dan ukuran blok dapat dipilih secara independen. Setiap blok dienkripsi dalam sejumlah putaran tertentu, sebagaimana halnya pada DES. Karena AES

menetapkan panjang kunci adalah 128, 192, dan 256, maka dikenal AES-128, AES-192, dan AES-256.

	Panjang Kunci (*Nk words*)	Ukuran Blok (*Nb words*)	Jumlah Putaran (*Nr*)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Catatan: 1 word = 32 bit

Secara de-fakto, hanya ada dua varian AES, yaitu AES-128 dan AES-256, karena akan sangat jarang pengguna menggunakan kunci yang panjangnya 192 bit. Karena AES mempunyai panjang kunci paling sedikit 128 bit, maka AES tahan terhadap serangan *exhaustive key search* dengan teknologi saat ini. Dengan panjang kunci 128-bit, maka terdapat sebanyak

$$2^{128} = 3{,}4 \times 10^{38}$$

kemungkinan kunci.

Jika digunakan komputer tercepat yang dapat mencoba 1 juta kunci setiap detik, maka akan dibutuhkan waktu $5{,}4 \times 10^{24}$ tahun untuk mencoba seluruh kemungkinan kunci. Jika digunakan komputer tercepat yang dapat mencoba 1 juta kunci setiap milidetik, maka akan dibutuhkan waktu $5{,}4 \times 10^{18}$ tahun untuk mencoba seluruh kemungkinan kunci.

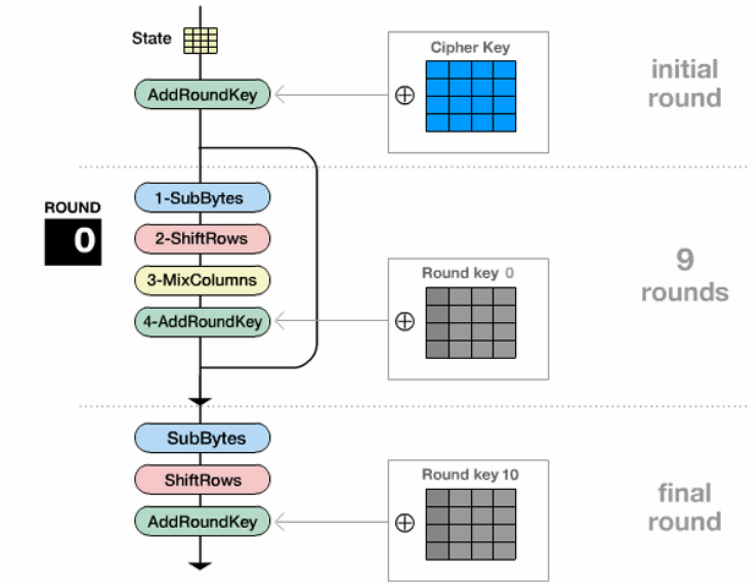
10.3. Algoritma Rijndael

Seperti pada DES, Rijndael menggunakan substitusi dan permutasi, dan sejumlah putaran (cipher berulang) setiap putaran mengunakan kunci internal yang berbeda (kunci setiap putaran disebut round key). Tetapi tidak seperti DES yang berorientasi bit,

Rijndael beroperasi dalam orientasi byte (untuk memangkuskan implementasi algoritma ke dalam *software* dan *hardware*).

Garis besar Algoritma Rijndael yang beroperasi pada blok 128-bit dengan kunci 128-bit adalah sebagai berikut (di luar proses pembangkitan round key):

1. AddRoundKey: melakukan XOR antara state awal (plainteks) dengan cipher key. Tahap ini disebut juga initial round.
2. Putaran sebanyak Nr – 1 kali. Proses yang dilakukan pada setiap putaran adalah:
   a. SubBytes: substitusi byte dengan menggunakan tabel substitusi (S-box).
   b. ShiftRows: pergeseran baris-baris array state secara wrapping.
   c. MixColumns: mengacak data di masing-masing kolom array state.
   d. AddRoundKey: melakukan XOR antara state sekarang round key.
3. Final round: proses untuk putaran terakhir:
   a. SubBytes
   b. ShiftRows
   c. AddRoundKey



Gambar 10.1 Diagram proses enkripsi

Versi 1: (putaran terakhir diperlakukan khusus)

```
#define LENGTH 16               /* Jumlah byte di dalam blok atau kunci */
#define NROWS 4                 /* Jumlah baris di dalam state */
#define NCOLS 4                 /* Jumlah kolom di dalam state */
#define ROUNDS 10               /* Jumlah putaran */
typedef unsigned char byte;  /* unsigned 8-bit integer */

rijndael (byte plaintext[LENGTH], byte ciphertext[LENGTH],
          byte key[LENGTH])
{
 int r;                        /* pencacah pengulangan */
 byte state[NROWS][NCOLS];  /* state sekarang */
 struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];  /* kunci pada
                                                   setiap putaran */

 KeyExpansion(key, rk);         /* bangkitkan kunci setiap putaran */
 CopyPlaintextToState(state, plaintext);  /* inisialisasi
                                             state sekarang */
 AddRoundKey(state, rk[0]);  /* XOR key ke dalam state */

 for (r = 1; r<= ROUNDS - 1; r++)
 {
  SubBytes(state);          /* substitusi setiap byte dengan S-box */
  ShiftRows(state);         /* rotasikan baris i sejauh i byte */
  MixColumns(state);    /* acak masing-masing kolom */
  AddRoundKey(state, rk[r]);   /* XOR key ke dalam state */
 }
 SubBytes(state);          /* substitusi setiap byte dengan S-box */
 ShiftRows(state);         /* rotasikan baris i sejauh i byte */
 AddRoundKey(state, rk[ROUNDS]);   /* XOR key ke dalam state */

 CopyStateToCiphertext(ciphertext, state);  /* blok cipherteks yang
                                               dihasilkan */

}
```

Versi 2: (proses pada setiap putaran sama)

```
#define LENGTH 16              /* Jumlah byte di dalam blok atau kunci */
#define NROWS 4                /* Jumlah baris di dalam state */
#define NCOLS 4                /* Jumlah kolom di dalam state */
#define ROUNDS 10              /* Jumlah putaran */
typedef unsigned char byte;    /* unsigned 8-bit integer */

rijndael (byte plaintext[LENGTH], byte ciphertext[LENGTH],
          byte key[LENGTH])
{
 int r;                       /* pencacah pengulangan */
 byte state[NROWS][NCOLS];    /* state sekarang */
 struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];   /* kunci pada
                                                    setiap putaran */

 KeyExpansion(key, rk);        /* bangkitkan kunci setiap putaran */
 CopyPlaintextToState(state, plaintext);  /* inisialisasi
                                             state sekarang */
 AddRoundKey(state, rk[0]);   /* XOR key ke dalam state */

 for (r = 1; r<= ROUNDS; r++)
 {
  SubBytes(state);          /* substitusi setiap byte dengan S-box */
  ShiftRows(state);         /* rotasikan baris i sejauh i byte */
  if (r < ROUNDS) MixColumns(state);   /* acak masing-masing kolom */
  AddRoundKey(state, rk[r]);    /* XOR key ke dalam state */
 }
 CopyStateToCiphertext(ciphertext, state);  /* blok cipherteks yang
                                               dihasilkan */

}
```

Algoritma Rijndael mempunyai 3 parameter:

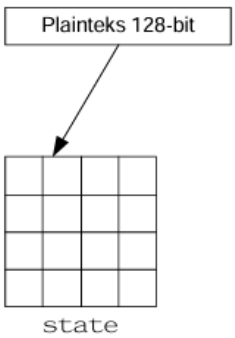
1. *plaintext* : array yang berukuran 16-byte, yang berisi data masukan.
2. *ciphertext* : array yang berukuran 16-byte, yang berisi hasil enkripsi.
3. *key* : array yang berukuran 16-byte, yang berisi kunci ciphering (disebut juga cipher key).

Dengan 16 byte, maka baik blok data dan kunci yang berukuran 128-bit dapat disimpan di dalam ketiga array tersebut (128 = 16 × 8).

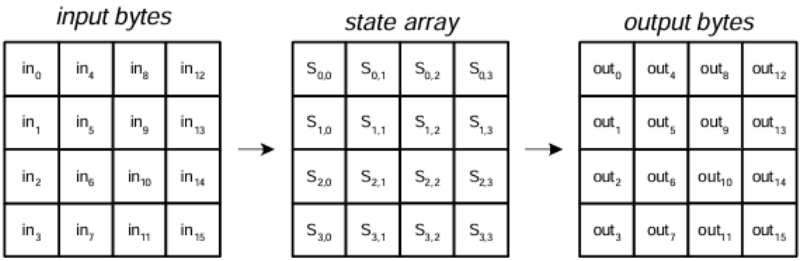
Selama kalkulasi plainteks menjadi cipherteks, status sekarang dari data disimpan di dalam array of bytes dua dimensi, state,

yang berukuran NROWS × NCOLS. Untuk blok data 128-bit, ukuran state adalah 4 × 4.

Elemen array state diacu sebagai S[r,c], dengan 0 ≤ r < 4 dan 0 ≤ c < Nb (Nb adalah panjang blok dibagi 32. Pada AES 128, Nb = 128/32 = 4).



Pada awal enkripsi, 16-byte data masukan, in0, in1, …, in15 disalin ke dalam array state (direalisasikan oleh fungsi CopyPlaintextToState(state, plaintext)) seperti diilustrasikan sebagai berikut:



Operasi enkripsi/dekripsi dilakukan terhadap array S, dan keluarannya ditampung didalam array out.

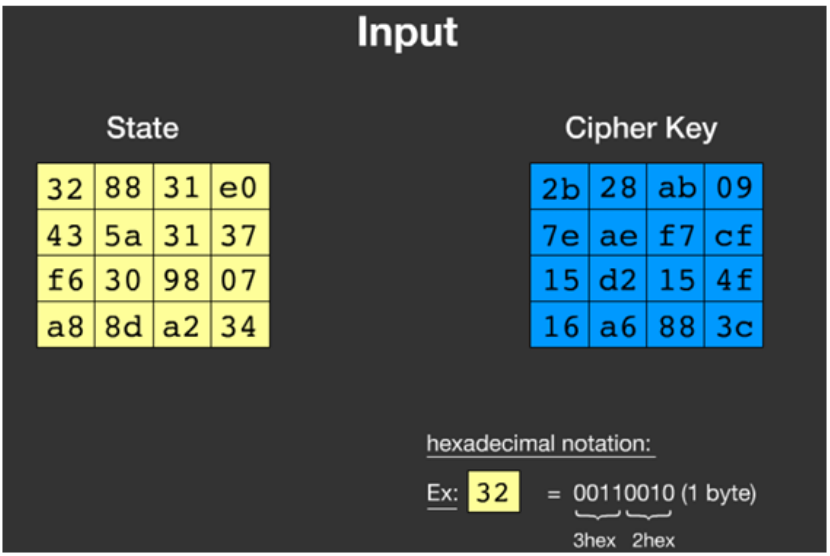
Skema penyalinan array masukan in ke array S:

$$S[r, c] \leftarrow in[r + 4c] \quad \text{untuk } 0 \le r < 4 \text{ dan } 0 \le c < Nb$$

Skema penyalinan array S ke array keluaran out:

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

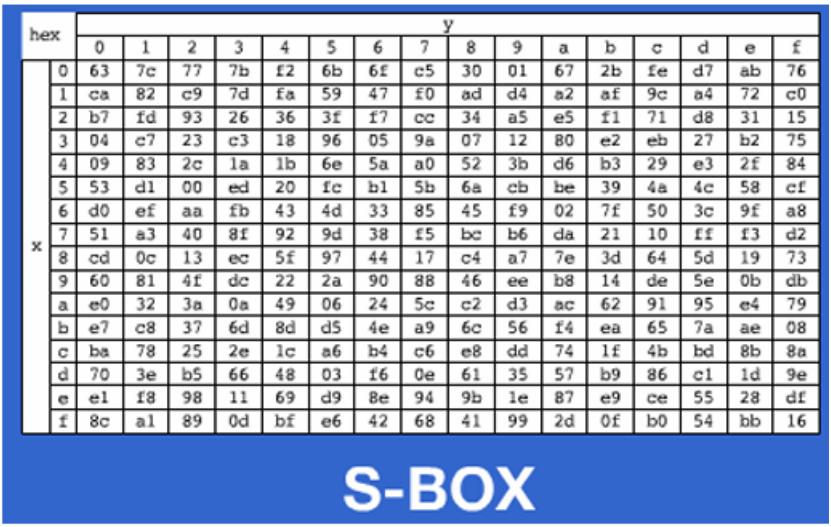
Contoh: (elemen state dan kunci dalam notasi HEX)



10.3.1. Transformasi SubBytes()

Transformasi SubBytes() memetakan setiap byte dari array state dengan menggunakan tabel substiusi S-box. Tidak seperti DES yang mempunyai S-box berbeda pada setiap putaran, AES hanya mempunyai satu buah S-box.

Tabel S-box yang digunakan adalah:



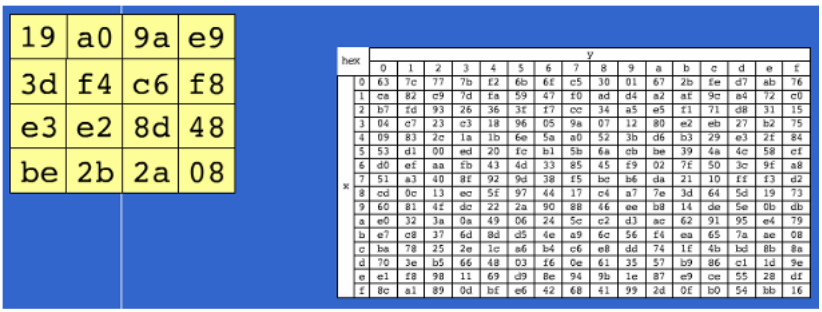
hex	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x 0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	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

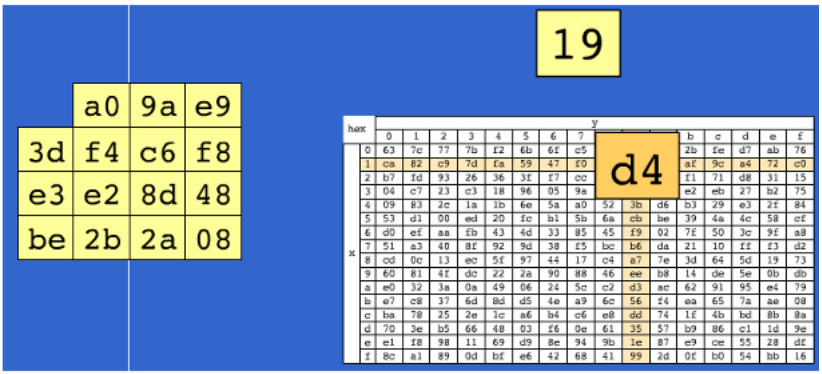
S-BOX

Cara pensubstitusian adalah sebagai berikut: untuk setiap byte pada array state, misalkan S[r, c] = xy, yang dalam hal ini xy adalah digit heksadesimal dari nilai S[r, c], maka nilai substitusinya, dinyatakan dengan S'[r, c], adalah elemen di dalam S-box yang merupakan perpotongan baris x dengan kolom y.

Misalnya S[0, 0] = 19, maka S'[0, 0] = d4

Contoh:



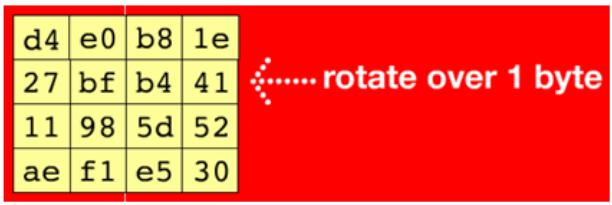


10.3.2. Transformasi ShiftRows()

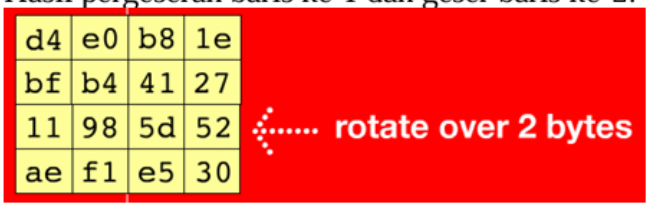
Transformasi ShiftRows() melakukan pergeseran secara wrapping (siklik) pada 3 baris terakhir dari array state. Jumlah pergeseran bergantung pada nilai baris (r). Baris r = 1 digeser sejauh 1 byte, baris r = 2 digeser sejauh 2 byte, dan baris r = 3 digeser sejauh 3 byte. Baris r = 0 tidak digeser.

Contoh :

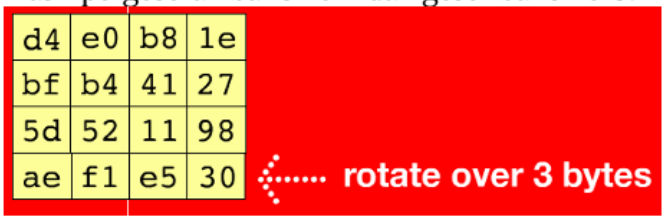
Geser baris ke-1:



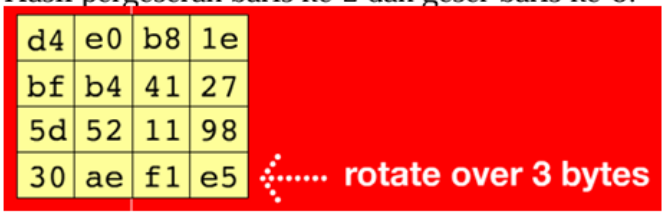
Hasil pergeseran baris ke-1 dan geser baris ke-2:



Hasil pergeseran baris ke-2 dan geser baris ke-3:



Hasil pergeseran baris ke-2 dan geser baris ke-3:



10.3.3. Transformasi MixColumns()

Transformasi MixColumns() mengalikan setiap kolom dari array state dengan polinom a(x) mod ($x^4 + 1$). Setiap kolom diperlakukan sebagai polinom 4-suku pada GF(2^8).

a(x) yang ditetapkan adalah:

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

Transformasi ini dinyatakan sebagai perkalian matriks:

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

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

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{3,c})$$

Contoh :

Hasil transformasi *ShiftRows()* sebelumnya:

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Operasi *MixColumns()* terhadap kolom pertama:

$$\begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} \bullet \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 02 & 03 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

Hasil transformasi *MixColumns()* seluruhnya:

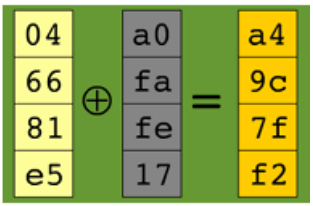
04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

10.3.4. Transformasi AddRoundKey()

Transformasi ini melakukan operasi XOR terhadap sebuah round key dengan array state, dan hasilnya disimpan di array state.

Contoh:

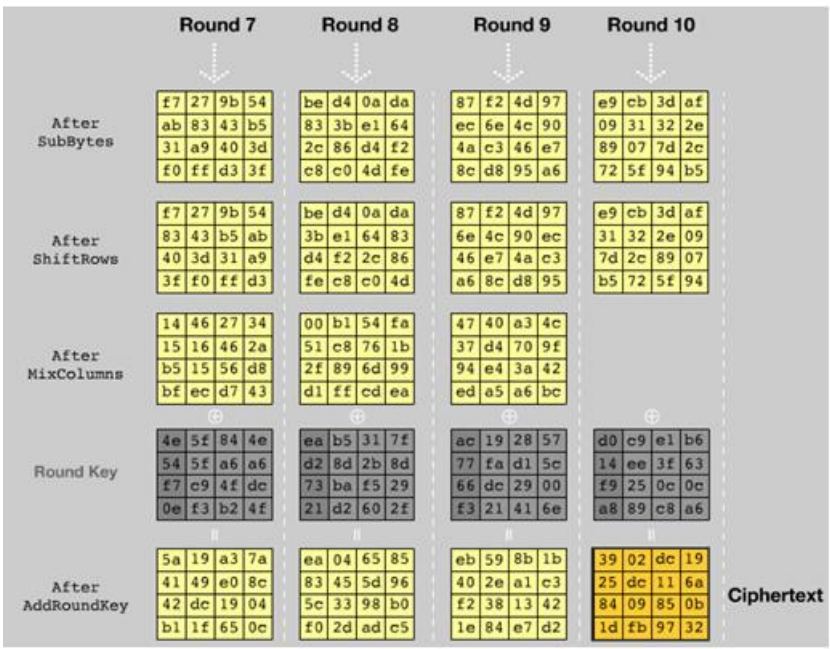
XOR-kan kolom pertama state dengan kolom pertama round key:



Hasil AddRoundKey() terhadap seluruh kolom:

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

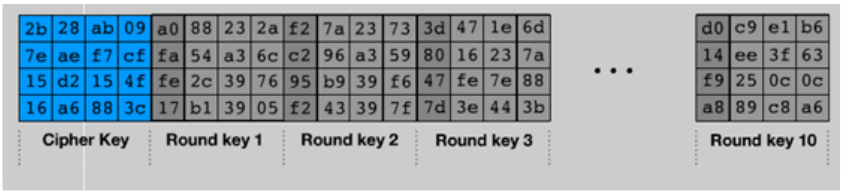

Round 2
Round 3
Round 4
Round 5
Round 6
After SubBytes
49 45 7f 77
de db 39 02
d2 96 87 53
89 f1 1a 3b
ac ef 13 45
73 c1 b5 23
cf 11 d6 5a
7b df b5 b8
52 85 e3 f6
50 a4 11 cf
2f 5e c8 6a
28 d7 07 94
e1 e8 35 97
4f fb c8 6c
d2 fb 96 ae
9b ba 53 7c
a1 78 10 4c
63 4f e8 d5
a8 29 3d 03
fc df 23 fe
After ShiftRows
49 45 7f 77
db 39 02 de
87 53 d2 96
3b 89 f1 1a
ac ef 13 45
c1 b5 23 73
d6 5a cf 11
b8 7b df b5
52 85 e3 f6
a4 11 cf 50
c8 6a 2f 5e
94 28 d7 07
e1 e8 35 97
fb c8 6c 4f
96 ae d2 fb
7c 9b ba 53
a1 78 10 4c
4f e8 d5 63
3d 03 a8 29
fe fc df 23
After MixColumns
58 1b db 1b
4d 4b e7 6b
ca 5a ca b0
f1 ac a8 e5
75 20 53 bb
ec 0b c0 25
09 63 cf d0
93 33 7c dc
0f 60 6f 5e
d6 31 c0 b3
da 38 10 13
a9 bf 6b 01
25 bd b6 4c
d1 11 3a 4c
a9 d1 33 c0
ad 68 8e b0
4b 2c 33 37
86 4a 9d d2
8d 89 f4 18
6d 80 e8 d8
Round Key
f2 7a 59 73
c2 96 35 59
95 b9 80 f6
f2 43 7a 7f
3d 47 1e 6d
80 16 23 7a
47 fe 7e 88
7d 3e 44 3b
ef a8 b6 db
44 52 71 0b
a5 5b 25 ad
41 7f 3b 00
d4 7c ca 11
d1 83 f2 f9
c6 9d b8 15
f8 87 bc bc
6d 11 db ca
88 0b f9 00
a3 3e 86 93
7a fd 41 fd
After AddRoundKey
aa 61 82 68
8f dd d2 32
5f e3 4a 46
03 ef d2 9a
48 67 4d d6
6c 1d e3 5f
4e 9d b1 58
ee 0d 38 e7
e0 c8 d9 85
92 63 b1 b8
7f 63 35 be
e8 c0 50 01
f1 c1 7c 5d
00 92 c8 b5
6f 4c 8b d5
55 ef 32 0c
26 3d e8 fd
0e 41 64 d2
2e b7 72 8b
17 7d a9 25


Round 7
Round 8
Round 9
Round 10
After SubBytes
f7 27 9b 54
ab 83 43 b5
31 a9 40 3d
f0 ff d3 3f
be d4 0a da
83 3b e1 64
2c 86 d4 f2
c8 c0 4d fe
87 f2 4d 97
ec 6e 4c 90
4a c3 46 e7
8c d8 95 a6
e9 cb 3d af
09 31 32 2e
89 07 7d 2c
72 5f 94 b5
After ShiftRows
f7 27 9b 54
83 43 b5 ab
40 3d 31 a9
3f f0 ff d3
be d4 0a da
3b e1 64 83
d4 f2 2c 86
fe c8 c0 4d
87 f2 4d 97
6e 4c 90 ec
46 e7 4a c3
a6 8c d8 95
e9 cb 3d af
31 32 2e 09
7d 2c 89 07
b5 72 5f 94
After MixColumns
14 46 27 34
15 16 46 2a
b5 15 56 d8
bf ec d7 43
00 b1 54 fa
51 c8 76 1b
2f 89 6d 99
d1 ff cd ea
47 40 a3 4c
37 d4 70 9f
94 e4 3a 42
ed a5 a6 bc
Round Key
4e 5f 84 4e
54 5f a6 a6
f7 c9 4f dc
0e f3 b2 4f
ea b5 31 7f
d2 8d 2b 8d
73 ba f5 29
21 d2 60 2f
ac 19 28 57
77 fa d1 5c
66 dc 29 00
f3 21 41 6e
d0 c9 e1 b6
14 ee 3f 63
f9 25 0c 0c
a8 89 c8 a6
After AddRoundKey
5a 19 a3 7a
41 49 e0 8c
42 dc 19 04
b1 1f 65 0c
ea 04 65 85
83 45 5d 96
5c 33 98 b0
f0 2d ad c5
eb 59 8b 1b
40 2e a1 c3
f2 38 13 42
1e 84 e7 d2
39 02 dc 19
25 dc 11 6a
84 09 85 0b
1d fb 97 32
Ciphertext

10.3.5. Ekspansi Kunci

Algoritma AES mengambil cipher key, K, yang diberikan oleh pengguna, dan memanggil fungsi KeyExpansion() untuk membangkitkan sejumlah *round key* (banyaknya round key bergantung pada jumlah putaran).

Contoh:



Proses pembangkitan kunci cukup kompleks, dan tidak dibahas di sini. URL yang terkait dengan AES:

1. AES Homepage, http://www.nist.gov/CryptoToolkit
2. J. Daemen, V. Rijmen, AES Proposal: Rijndael, http://www.esat.kuleuven.ac.be/~rizmen/

Beberapa algoritma kriptografi simetri:

Cipher	Pembuat	Panjang Kunci	Keterangan
Blowfish	Bruce Schneier	1 – 448 bit	*Old and slow*
DES	IBM	56 bit	*Too weak to use now*
IDEA	Massey dan Xuejia	128 bit	*Good, but patented*
RC4	Ronald Rivest	1 – 2048 bit	*Caution: some keys are weak*
RC5	Ronald Rivest	128 – 256 bit	*Good, but patented*
Rijndael	Daemen dan Rijmen	128 – 256 bit	*Best choice*
Serpent	Anderson, Biham, Knudsen	128 – 256 bit	*Very strong*
Triple DES	IBM	168 bit	*Second best choice*
Twofish	Bruce Schneier	128 – 256 bit	*Very strong; widely used*

11.
Sistem Kriptografi **Kunci-Publik**

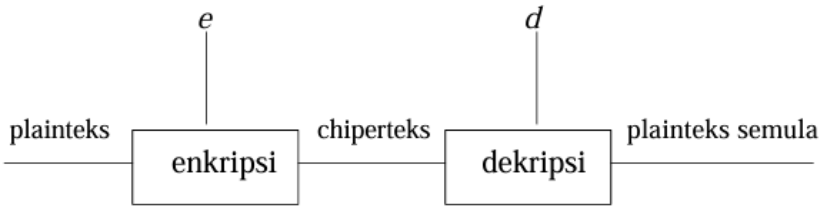
11.1. Pendahuluan

Sampai akhir tahun 1970, hanya ada sistem kriptografi simetri. Karena sistem kriptografi simetri menggunakan kunci yang sama untuk enkripsi dan dekripsi, maka hal ini mengimplikasikan dua pihak yang berkomunikasi saling mempercayai. Kedua pihak harus menjaga kerahasiaan kunci (sehingga, junci enkripsi/dekripsi disebut juga secret key)

Pada sistem kriptografi kunci-publik, kunci kriptografi dibuat sepasang, satu kunci untuk enkripsi dan satu kunci untuk dekripsi (Gambar 11.1);

- Kunci untuk enkripsi diumumkan kepada publik oleh karena itu tidak rahasia sehingga dinamakan kunci publik (public-key), disimbolkan dengan e.
- Kunci untuk dekripsi bersifat rahasia sehingga dinamakan kunci privat (private key), disimbolkan dengan d.

Karena ada kunci enkripsi ≠ kunci dekripsi, maka sistem kriptografi kunci-publik kadang-kadang disebut juga sistem kriptografi asimetri.



Gambar 11.1 Sistem kriptografi kunci-publik

Ket: e = public key, d = private key

Sistem kriptografi kunci-publik didasarkan pada fakta:
1. Komputasi untuk enkripsi/dekripsi pesan mudah dilakukan.
2.Secara komputasi hampir tidak mungkin (infeasible) menurunkan kunci privat, d, bila diketahui kunci publik, e, pasangannya.

Kedua fakta di atas analog dengan:

- Perkalian vs pemfaktoran
  Mengalikan dua buah bilangan prima, a × b = n, mudah, tetapi memfaktorkan n menjadi faktor-faktor primanya sulit.
  Contoh: 31 × 47 = 1457 (perkalian)
  1457 = ? × ? (pemfaktoran)

- Perpangkatan vs logaritmik
  Melakukan perpangkatan, $y = a^x$, mudah, tetapi menghitung $x = {}^a\log y$ sulit jika a tidak diketahui.
  Contoh: 125 = 248832 (perpangkatan)
  x = a log 248832 = ? (logaritmik)

11.2. Konsep Kriptografi Kunci-Publik

Konsep kriptografi kunci-publik sederhana dan elegan, tetapi mempunyai konsekuensi penggunaan yang hebat. Misalkan *E* adalah fungsi enkripsi dan *D* adalah fungsi dekripsi. Misalkan (*e, d*) adalah pasangan kunci untuk enkripsi dan dekripsi sedemikian sehingga

$$E_d(m) = c \text{ dan } D_d(c) = m$$

untuk suatu plainteks *m* dan cipherteks *c*.

Kedua persamaan ini menyiratkan bahwa dengan mengetahui *e* dan *c*, maka secara komputasi hampir tidak mungkin menemukan *m*. Asumsi lainnya, dengan mengetahui e,

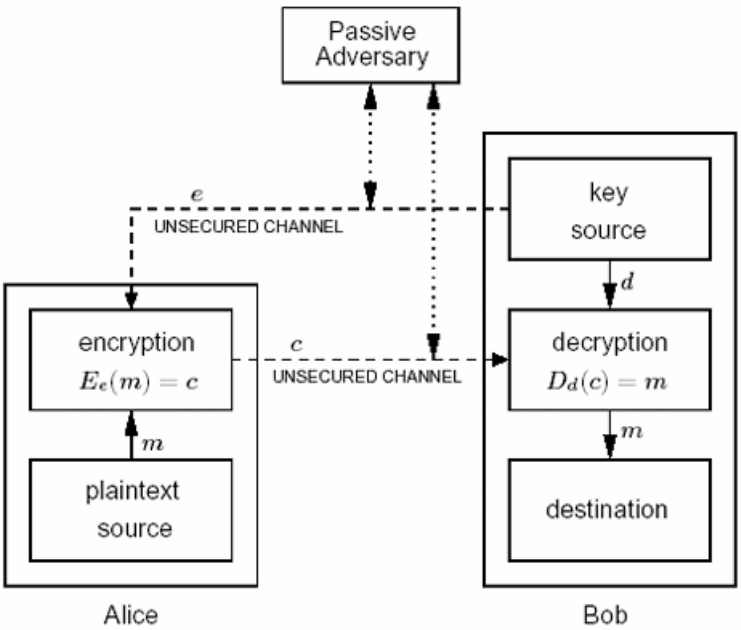
secara komputasi hampir tidak mungkin menurunkan d. E_e digambarkan sebagai fungsi pintu-kolong (trapdoor) satu arah dengan d adalah informasi *trapdoor* yang diperlukan untuk menghitung fungsi inversinya, D, yang dalam hal ini membuat proses dekripsi dapat dilakukan.

Konsep di atas menjadi penting bila kriptografi kunci-publik digunakan untuk mengamankan pertukaran pesan dari dua entitas yang berkomunikasi.

Misalkan Alice berkomunikasi dengan Bob. Bob memilih pasangan kunci (e, d). Bob mengirimkan kunci enkripsi e (kunci publik) kepada Alice melalui sembarang saluran tetapi tetap menjaga kerahasiaan kunci dekripsinya, d (kunci privat).

Kemudian, Alice ingin mengirim pesan m kepada Bob. Alice mengenkripsikan pesan m dengan menggunakan kunci publik Bob, untuk mendapatkan $c = E_e(m)$, lalu mengirimkan c melalui saluran komunikasi (yang tidak perlu aman). Bob mendekripsi cipherteks c dengan menggunakan kunci privatnya untuk memperoleh $m = D_d(c)$.

Perhatikan skema komunikasi dengan kriptografi kunci publik pada Gambar 11.2. Gambar ini memperlihatkan perbedaan mendasar sistem asimetri dengan sistem simetri. Di sini kunci enkripsi dikirim kepada Alice melalui saluran yang tidak perlu aman (*unsecure channel*). Saluran yang tidak perlu aman ini mungkin sama dengan saluran yang digunakan untuk mengirim cipherteks.



Gambar 11.2 Enkripsi/dekripsi dengan kriptorafi kunci-publik.

Sistem kriptografi kunci-publik juga cocok untuk kelompok pengguna di lingkungan jaringan komputer (LAN/WAN). Setiap pengguna jaringan mempunyai pasangan kunci publik dan kunci privat yang bersuaian. Kunci publik, karena tidak rahasia, biasanya disimpan di dalam basisdata kunci yang dapat diakses oleh pengguna lain. Jika ada pengguna yang hendak berkirim pesan ke pengguna lainnya, maka ia ia perlu mengetahui kunci publik penerima pesan melalui basisdata kunci ini lalu menggunakannya untuk mengenkripsi pesan. Hanya penerima pesan yang berhak yang dapat mendekripsi pesan karena ia mempunyai kunci privat.

Dengan sistem kriptografi kunci-publik, tidak diperlukan pengiriman kunci privat melalui saluran komunikasi khusus sebagaimana pada sistem kriptografi simetri. Meskipun kunci publik diumumkan ke setiap orang di dalam kelompok, namun

kunci publik perlu dilindungi agar otentikasinya terjamin (misalnya tidak diubah oleh orang lain).

11.3. Kriptografi Simetri vs Kriptografi Asimetri

Baik kriptografi simetri maupun kriptografi asimetri (kunci publik), keduanya mempunyai kelebihan dan kelemahan.

Kelebihan kriptografi simetri:

1. Algoritma kriptografi simetri dirancang sehingga proses enkripsi/dekripsi membutuhkan waktu yang singkat.
2. Ukuran kunci simetri relatif pendek.
3. Algoritma kriptografi simetri dapat digunakan untuk membangkitkan bilangan acak.
   Algorima kriptografi simetri dapat disusun untuk menghasilkan cipher yang lebih kuat.
4. Otentikasi pengirim pesan langsung diketahui dari cipherteks yang diterima, karena kunci hanya diketahui oleh pengirim dan penerima pesan saja.

Kelemahan kriptografi simetri:

1. Kunci simetri harus dikirim melalui saluran yang aman. Kedua entitas yang berkomunikasi harus menjaga kerahasisan kunci ini.
2. Kunci harus sering diubah, mungkin pada setiap sesi komunikasi.

Kelebihan kriptografi kunci-publik (asimetri):

1. Hanya kunci privat yang perlu dijaga kerahasiaannya oleh seiap entitas yang berkomuniaksi (tetapi, otentikasi kunci publik tetap harus terjamin). Tidak ada kebutuhan mengirim kunci kunci privat sebagaimana pada sistem simetri.

2. Pasangan kunci publik/kunci privat tidak perlu diubah, bahkan dalam periode waktu yang panjang.
3. Dapat digunakan untuk mengamankan pengiriman kunci simetri.
5. Beberapa algoritma kunci-publik dapat digunakan untuk memberi tanda tangan digital pada pesan (akan dijelaskan pada materi kuliah selanjutnya)

Kelemahan kriptografi kunci-publik (asimetri):

1. Enkripsi dan dekripsi data umumnya lebih lambat daripada sistem simetri, karena enkripsi dan dekripsi menggunakan bilangan yang besar dan melibatkan operasi perpangkatan yang besar.
2. Ukuran cipherteks lebih besar daripada plainteks (bisa dua sampai empat kali ukuran plainteks).
3. Ukuran kunci relatif lebih besar daripada ukuran kunci simetri.
4. Karena kunci publik diketahui secara luas dan dapat digunakan setiap orang, maka cipherteks tidak memberikan informasi mengenai otentikasi pengirim.
6. Tidak ada algoritma kunci-publik yang terbukti aman (sama seperti block cipher). Kebanyakan aalgoriam mendasakan keamanannya pada sulitnya memecahkan persoalan-persoalan aritmetik (pemfaktoran, logaritmik, dsb) yang menjadi dasar pembangkitan kunci.

11.4. Aplikasi Kriptografi Knci-Publik

Aplikasi kriptogafi kunci-publik dapat dibagi menjadi 3 kategori:

1. Enkripsi/dekripsi
   Seperti pada algoritma kriptografi simetri, algoritma kunci-publik dapat digunakan untuk menjaga kerahasiaan pesan (provide confidentiality/secrecy).

Contoh algoritma: RSA, Knapsack, Rabin, ElGamal

2. Digital signatures
   Algoritma kriptografi kunci-publik dapat digunakan untuk membuktikan otentikasi pesan maupun otentikasi pengirim (provide authentication)
   Contoh algoritma: RSA, DSA, ElGamal, GOST

3. Pertukaran kunci (key exchange)
   Algoritma kriptografi kunci-publik dapat digunakan untuk pengiriman kunci simetri (session keys)
   Contoh algoritma: RSA, Diffie-Hellman

Beberapa algoritma kriptografi kunci-publik cocok digunakan untuk ketiga macam kategori aplikasi (misalnya RSA), beberapa algoritma hanya ditujukan untuk aplikasi spesifik (misalya DSA untuk digital signature).

12.
Algoritma RSA dan ElGamal

12.1. Pendahuluan

Dari sekian banyak algoritma kriptografi kunci-publik yang pernah dibuat, algoritma yang paling populer adalah algoritma RSA. Algoritma RSA dibuat oleh 3 orang peneliti dari MIT (Massachussets Institute of Technology) pada tahun 1976, yaitu: Ron (R)ivest, Adi (S)hamir, dan Leonard (A)dleman.

Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor prima. Pemfaktoran dilakukan untuk memperoleh kunci privat. Selama pemfaktoran bilangan besar menjadi faktor faktor prima belum ditemukan algoritma yang mangkus, maka selama itu pula keamanan algoritma RSA tetap terjamin.

12.2. Properti Algoritma RSA

Besaran-besaran yang digunakan pada algoritma 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)
5. d (kunci dekripsi)	(rahasia)
6. m (plainteks)	(rahasia)
7. c (cipherteks)	(tidak rahasia)

12.3. Perumusan Algoritma RSA

Algoritma RSA didasarkan pada teorema Euler (lihat bahan kuliah Teori Bilangan) yang menyatakan bahwa

$$a^{f(n)} \equiv 1 \pmod{n} \qquad \textbf{(12.1)}$$

yang dalam hal ini,

1. *a* harus relatif prima terhadap *n*.
2. f*(n) = n(1 – 1/p1)(1 – 1/p2) … (1 – 1/pr)*, yang dalam hal ini *p1, p2, …, pr* adalah faktor prima dari *n*.

f(n) adalah fungsi yang menentukan berapa banyak dari bilangan-bilangan 1, 2, 3, …, *n* yang relatif prima terhadap *n*.
Berdasarkan sifat $a^k \equiv b^k \pmod{n}$ untuk *k* bilangan bulat ≥ 1, maka persamaan (12.1) dapat ditulis menjadi

$$a^{k\phi(n)} \equiv 1^k \pmod{n}$$

atau

$$a^{k\phi(n)} \equiv 1 \pmod{n} \qquad \textbf{(12.2)}$$

Bila *a* diganti dengan *m*, maka persamaan (12.2) menjadi

$$m^{k\phi(n)} \equiv 1 \pmod{n} \qquad \textbf{(12.3)}$$

Berdasarkan sifat ac ≡ bc (mod n), maka bila persamaan (1.3) dikali dengan m menjadi:

$$m^{k\phi(n)+1} \equiv m \pmod{n} \qquad \textbf{(12.4)}$$

yang dalam hal ini m relatif prima terhadap n.
Misalkan *e* dan *d* dipilih sedemikian sehingga

$$e \cdot d \equiv 1 \pmod{\phi(n)} \qquad \textbf{(12.5)}$$

Atau

$$e \cdot d = k\phi(n) + 1 \qquad \textbf{(12.6)}$$

Sulihkan (12.6) ke dalam persamaan (12.4) menjadi:

$$m^{e\ d} \equiv m \pmod{n} \qquad \textbf{(12.7)}$$

Persamaan (12.7) dapat ditulis kembali menjadi

$$(m^{e})^{d} \equiv m \pmod{n} \qquad \textbf{(12.8)}$$

yang artinya, perpangkatan m dengan e diikuti dengan perpangkatan dengan d menghasilkan kembali m semula.

Berdasarkan persamaan (12.8), maka enkripsi dan dekripsi dirumuskan sebagai berikut:

$$E_e(m) = c \equiv m^e \bmod n \qquad \textbf{(12.9)}$$

$$D_d(c) = m \equiv c^d \bmod n \qquad \textbf{(12.10)}$$

Karena $e \cdot d = d \cdot e$, maka enkripsi diikuti dengan dekripsi ekivalen dengan dekripsi diikuti enkripsi:

$$D_d(E_e(m)) = E_e(D_d(m)) \equiv m^d \bmod n \qquad \textbf{(12.11)}$$

Oleh karena $m^d \bmod n \equiv (m + jn)^d \bmod n$ untuk sembarang bilangan bulat j, maka tiap plainteks m, $m + n$, $m + 2n$, …, menghasilkan cipherteks yang sama. Dengan kata lain, transformasinya dari banyak ke satu.

Agar transformasinya satu-ke-satu, maka m harus dibatasi dalam himpunan {0, 1, 2, …, n – 1} sehingga enkripsi dan dekripsi tetap benar seperti pada persamaan (12.8) dan (12.9).

12.4. Algoritma Membangkitkan Pasangan Kunci

1. Pilih dua buah bilangan prima sembarang, p dan q.

2. Hitung $n = p \cdot q$ (sebaiknya $p \neq q$, sebab jika $p = q$ maka $n = p^2$ sehingga p dapat diperoleh dengan menarik akar pangkat dua dari n).
3. Hitung $\phi(n) = (p - 1)(q - 1)$.
4. Pilih kunci publik, e, yang relatif prima terhadap $\phi(n)$.
5. Bangkitkan kunci privat dengan menggunakan persamaan (12.5), yaitu $e \cdot d \equiv 1 \pmod{\phi(n)}$.

Perhatikan bahwa $e \cdot d \equiv 1 \pmod{\phi(n)}$ ekivalen dengan $e \cdot d = 1 + k\phi(n)$, sehingga d dapat dihitung dengan

$$d = \frac{1 + k\phi(n)}{e} \qquad \textbf{(12.12)}$$

Akan terdapat bilangan bulat k yang memberikan bilangan bulat d.

Hasil dari algoritma di atas:

- Kunci publik adalah pasangan (e, n)
- Kunci privat adalah pasangan (d, n)

Catatan: n tidak bersifat rahasia, namun ia diperlukan pada perhitungan enkripsi/dekripsi.

Contoh 12.1. Misalkan A akan membangkitkan kunci publik dan kunci privat miliknya. A memilih $p = 47$ dan $q = 71$ (keduanya prima).
Selanjutnya, A menghitung

$$n = p \cdot q = 3337$$

dan

$$\phi(n) = (p - 1)(q - 1) = 3220.$$

A memilih kunci publik $e = 79$, karena 79 relatif prima dengan 3220. A mengumumkan nilai e dan n.

Selanjutnya *A* menghitung kunci dekripsi *d* seperti yang dituliskan pada langkah instruksi 5 dengan menggunakan persamaan (12.12),

$$d = \frac{1 + (k \times 3220)}{79}$$

Dengan mencoba nilai-nilai k = 1, 2, 3, …, diperoleh nilai *d* yang bulat adalah 1019. Ini adalah kunci privat untuk mendekripsi pesan. Kunci ini harus dirahasiakan oleh *A*.

Kunci publik: (e = 79, n = 3337)
Kunci privat: (d = 1019, n = 3337)

12.5. Algoritma Enkripsi/Dekripsi

Enkripsi

1. Ambil kunci publik penerima pesan, *e*, dan modulus *n*.
2. Nyatakan plainteks *m* menjadi blok-blok *m1, m2,* …, sedemikian sehingga setiap blok merepresentasikan nilai di dalam selang [0, n – 1].
3. Setiap blok mi dienkripsi menjadi blok c_i dengan rumus

$$m_i = c_i^{\,d} \bmod n$$

Dekripsi

1. Setiap blok cipherteks c_i didekripsi kembali menjadi blok m_i dengan rumus $m_i = c_i^{\,d} \bmod n$.

Contoh 12.2. Misalkan *B* mengirim pesan kepada *A*. Pesan (plainteks) yang akan dikirim oleh *A* adalah

$$m = \text{HARI INI}$$

atau dalam sistem desimal (pengkodean ASCII) adalah

7265827332737873

B memecah m menjadi blok yang lebih kecil, misalnya m dipecah menjadi enam blok yang berukuran 3 digit:

$$m_1 = 726 \qquad m_4 = 273$$
$$m_2 = 582 \qquad m_5 = 787$$
$$m_3 = 733 \qquad m_6 = 003$$

Nilai-nilai m_i ini masih terletak di dalam selang [0, 3337 – 1] agar transformasi menjadi satu-ke-satu.
B mengetahui kunci publik *A* adalah $e = 79$ dan $n = 3337$. *A* dapat mengenkripsisikan setiap blok plainteks sebagai berikut:

$$c_1 = 726^{79} \bmod 3337 = 215; \qquad c_2 = 582^{79} \bmod 3337 = 776;$$
$$c_3 = 733^{79} \bmod 3337 = 1743; \qquad c_4 = 273^{79} \bmod 3337 = 933;$$
$$c_5 = 787^{79} \bmod 3337 = 1731; \qquad c_6 = 003^{79} \bmod 3337 = 158$$

Jadi, cipherteks yang dihasilkan adalah

$$c = 215\ 776\ 1743\ 933\ 1731\ 158.$$

Dekripsi dilakukan dengan menggunakan kunci privat

$$d = 1019$$

Blok-blok cipherteks didekripsikan sebagai berikut:

$$m_1 = 215^{1019} \bmod 3337 = 726$$
$$m_2 = 776^{1019} \bmod 3337 = 582$$
$$m_3 = 1743^{1019} \bmod 3337 = 733$$
$$\ldots$$

Blok plainteks yang lain dikembalikan dengan cara yang serupa. Akhirnya kita memperoleh kembali plainteks semula

$$m = 7265827332737873$$

yang dalam sistem pengkodean ASCII adalah

$$m = \text{HARI INI}.$$

12.6. Keamanan RSA

Kamanan algoritma RSA didasarkan pada sulitnya memfaktorkan bilangan besar menjadi fakto-faktor primanya.

Masalah pemfaktoran: Faktorkan n, yang dalam hal ini n adalah hasil kali dari dua atau lebih bilangan prima.

Pada RSA, masalah pemfaktoran berbunyi: Faktorkan n menjadi dua faktor primanya, p dan q, sedemikian sehingga $n = p \cdot q$.

Sekali n berhasil difaktorkan menjadi p dan q, maka $\phi(n) = (p - 1)(q-1)$ dapat dihitung. Selanjutnya, karena kunci enkrispi e diumumkan (tidak rahasia), maka kunci dekripsi d dapat dihitung dari persamaan $e \cdot d \equiv 1 \pmod{\phi(n)}$.

Penemu algoritma RSA menyarankan nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali $n = p \times q$ akan berukuran lebih dari 200 digit.

Menurut Rivest dan kawan-kawan, usaha untuk mencari faktor prima dari bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun, sedangkan untuk bilangan 500 digit membutuhkan waktu 10^{25} tahun! (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik).

Untunglah algoritma yang paling mangkus untuk memfaktorkan bilangan yang besar belum ditemukan. Selama 300 tahun para matematikawan mencoba mencari faktor bilangan yang besar namun tidak banyak membuahkan hasil. Semua bukti yang diketahui menunjukkan bahwa upaya pemfaktoran itu lura biasa sulit.

Fakta inilah yang membuat algoritma RSA tetap dipakai hingga saat ini. Selagi belum ditemukan algoritma yang mangkus untuk memfaktorkan bilangan bulat menjadi faktor primanya, maka algoritma RSA tetap direkomendasikan untuk

mengenkripsi pesan. Untuk informasi RSA lebih lanjut, kunjungi situs web *www.rsasecurity.com*

12.7. Algoritma ElGamal

Algoritma ElGamal juga adalah algoritma kriptografi kunci publik. Algoritma ini pada mulanya digunakan untuk digital signature, namun kemudian dimodifikasi sehingga juga bisa digunakan untuk enkripsi dan dekripsi. Keamanan algoritma ini terletak pada sulitnya menghitung logaritma diskrit. Masalah logaritma diskrit: Jika p adalah bilangan prima dan g dan y adalah sembarang bilangan bulat. Carilah x sedemikian sehingga $g^x \equiv y \pmod{p}$

Besaran-besaran yang digunakan di dalam algoritma ElGamal adalah:

1. Bilangan prima, p (tidak rahasia)
2. Bilangan acak, g ($g < p$) (tidak rahasia)
3. Bilangan acak, x ($x < p$) (rahasia, kunci privat)
4. $y = g^x \bmod p$ (tidak rahasia, k. publik)
5. m (plainteks) (rahasia)
6. a dan b (cipherteks (tidak rahasia)

Algoritma Membangkitkan Pasangan Kunci

1. Pilih sembarang bilangan prima p (p dapat di-share di antara anggota kelompok)
2. Pilih dua buah bilangan acak, g dan x, dengan syarat $g < p$ dan $1 \le x \le p - 2$
3. Hitung $y = g^x \bmod p$.

Algoritma Enkripsi/Dekripsi

Kunci publik: tripel (y, g, p)

Kunci privat: pasangan (x, p)

Catatan: g dan p tidak rahasia, namun diperlukan pada perhitungan enkripsi/dekripsi (lihat algoritma enkripsi/dekripsi di bawah).

Algoritma Enkripsi/Dekripsi

Enkripsi:

1. Susun plainteks menjadi blok-blok *m1, m2*, …, sedemikian sehingga setiap blok merepresentasikan nilai di dalam selang [0, p – 1].
2. Pilih bilangan acak k, yang dalam hal ini $1 \le k \le p - 2$.
3. Setiap blok m dienkripsi dengan rumus
   $a = g^k \bmod p$
   $b = y^k m \bmod p$
   Pasangan a dan b adalah cipherteks untuk blok pesan m. Jadi, ukuran cipherteks dua kali ukuran plainteksnya.

Dekripsi

Gunakan kunci privat x untuk mendekripsi a dan b menjadi plainteks m dengan persamaan:

$$m = b/a^x \bmod p$$

$$\text{(Catatan: } (a^x)^{-1} = a^{p-1-x} \bmod p \text{)}$$

(Catatlah bahwa karena $a^x \equiv g^{kx} \pmod p$ maka

$$\begin{aligned} b/a^x &\equiv y^k m/a^x \\ &\equiv g^{xk} m/g^{xk} \\ &\equiv m \pmod p, \end{aligned}$$

yang berarti bahwa plainteks dapat ditemukan kembali dari pasangan cipherteks a dan b)

Contoh 15.3. Misalkan A ingin membangkitkan pasangan kuncinya. A memilih p = 2357, g = 2, dan x = 1751.
A menghitung
$y = g^x \bmod p = 2^{1751} \bmod 2357 = 1185$

Jadi, kunci publik A adalah (y = 1185, g = 2, p = 2357) dan kunci privatnya adalah (x = 1751, p = 2357).

Enkripsi: Misalkan *B* ingin mengirim plainteks m = 2035 kepada *A* (nilai m masih berada di dalam selang [0, 2357 – 1]).
B memilih bilangan acak k = 1520 (nilai k masih berada di dalam selang [0, 2357 – 1]).

B menghitung

$$a = g^k \bmod p = 2^{1520} \bmod 2357 = 1430$$
$$b = y^k m \bmod p = 1185^{1520} \cdot 2035 \bmod 2357 = 697$$

Jadi, cipherteks yang dihasilkan adalah (1430, 697). B mengirim cipherteks ini ke A.
Dekripsi: A mendekripsi cipherteks dari B dengan melakukan perhitungan sebagai berikut:

$$(a^x)^{-1} = a^{p-1-x} \bmod p = 1430^{605} \bmod 2357 = 872$$

$$m = b/a^x \bmod p = 697 \cdot 872 \bmod 2357 = 2035$$

Paten

Algoritma ElGamal tidak dipatenkan. Tetapi, algoritma ini didasarkan pada algoritma Diffie-Hellman, sehingga hak paten algoritam Diffie-Hellman juga mencakup algoritma ElGamal. Untunglah hak paten algoritma Diffie-Hellman berakhir pada bulan April 1997, sehingga algoritma ElGamal dapat diimplementasikan untuk aplikasi komersil.

13.
Algoritma **Knapsack**

13.1. Algoritma ElGamal

Algoritma Knapsack juga adalah algoritma kriptografi kunci publik. Keamanan algoritma ini terletak pada sulitnya memecahkan persoalan knapsack (Knapsack Problem). Knapsack artinya karung/kantung. Karung mempunyai kapasitas muat terbatas. Barang-barang dimasukkan ke dalam karung hanya sampai batas kapasitas maksimum karung saja.

Knapsack Problem:

Diberikan bobot knapsack adalah *M*. Diketahui *n* buah objek yang masing-masing bobotnya adalah *w1, w2, …, wn*. Tentukan nilai bi sedemikian sehingga

$$M = b_1 w_1 + b_2 w_2 + \ldots + b_n w_n \qquad \textbf{(13.1)}$$

yang dalam hal ini, b_i bernilai 0 atau 1. Jika $b_i = 1$, berarti objek i dimasukkan ke dalam knapsack, sebaliknya jika $b_i = 0$, objek *i* tidak dimasukkan.

Dalam teori algoritma, persoalan knapsack termasuk ke dalam kelompok *NP-complete*. Persoalan yang termasuk *NP complete* tidak dapat dipecahkan dalam orde waktu polinomial.

13.2. Algoritma Knapsack Sederhana

Ide dasar dari algoritma kriptografi knapsack adalah mengkodekan pesan sebagai rangkaian solusi dari dari persoalan knapsack. Setiap bobot wi di dalam persoalan knapsack merupakan kunci privat, sedangkan bit-bit plainteks menyatakan b_i.

Contoh 13.1. Misalkan $n = 6$ dan $w_1 = 1$, $w_2 = 5$, $w_3 = 6$, $w_4 = 11$, $w_5 = 14$, dan $w_6 = 20$.

Plainteks: 111001010110000000011000

Plainteks dibagi menjadi blok yang panjangnya n, kemudian setiap bit di dalam blok dikalikan dengan w_i yang berkorepsonden sesuai dengan persamaan (1):

Blok plainteks ke-1	: 111001
Knapsack	: 1, 5, 6, 11, 14, 20
Kriptogram	: $(1 \times 1) + (1 \times 5) + (1 \times 6) + (1 \times 20)$ $= 32$
Blok plainteks ke-2	: 010110
Knapsack	: 1, 5, 6, 11, 14, 20
Kriptogram	: $(1 \times 5) + (1 \times 11) + (1 \times 14) = 30$
Blok plainteks ke-3	: 000000
Knapsack	: 1, 5, 6, 11, 14, 20
Kriptogram	: 0
Blok plainteks ke-4	: 011000
Knapsack	: 1, 5, 6, 11, 14, 20
Kriptogram	: $(1 \times 5) + (1 \times 6) = 11$

Jadi, cipherteks yang dihasilkan: 32 30 0 11

Sayangnya, algoritma knapsack sederhana di atas hanya dapat digunakan untuk enkripsi, tetapi tidak dirancang untuk dekripsi.

Misalnya, jika diberikan kriptogram = 32, maka tentukan b1, b2, …, b6 sedemikian sehingga

$$32 = b_1 + 5b_2 + 6b_3 + 11b_4 + 14b_5 + 20b_6 \qquad (2)$$

Solusi persamaan (2) ini tidak dapat dipecahkan dalam orde waktu polinomial dengan semakin besarnya n (dengan catatan barisan bobot tidak dalam urutan menaik). Namun, hal inilah yang dijadikan sebagai kekuatan algoritma knapsack.

13.3. Superincreasing Knapsack

Superincreasing knapsack adalah persoalan knapsack yang dapat dipecahkan dalam orde $O(n)$ (jadi, polinomial). Ini adalah persoalan knapsack yang mudah sehingga tidak disukai untuk dijadikan sebagai algoritma kriptografi yang kuat.

Jika senarai bobot disebut barisan superincreasing, maka kita dapat membentuk superincreasing knapsack. Barisan superincreasing adalah suatu barisan di mana setiap nilai di dalam barisan lebih besar daripada jumlah semua nilai sebelumnya.

Misalnya {1, 3, 6, 13, 27, 52} adalah barisan superincreasing, tetapi {1, 3, 4, 9, 15, 25} bukan.

Solusi dari superincreasing knapsack (yaitu $b_1, b_2, \ldots, b_n$) mudah dicari sebagai berikut (berarti sama dengan mendekripsikan cipherteks menjadi plainteks semula):

1. Jumlahkan semua bobot di dalam barisan.
2. Bandingkan bobot total dengan bobot terbesar di dalam barisan. Jika bobot terbesar lebih kecil atau sama dengan bobot total, maka ia dimasukkan ke dalam knapsack, jika tidak, maka ia tidak dimasukkan.
3. Kurangi bobot total dengan bobot yang telah dimasukkan, kemudian bandingkan bobot total sekarang dengan bobot

terbesar selanjutnya. Demikian seterusnya sampai seluruh bobot di dalam barisan selesai dibandingkan.

4. Jika bobot total menjadi nol, maka terdapat solusi persoalan superincreasing knapsack , tetapi jika tidak nol, maka tidak ada solusinya.

Contoh 13.2. Misalkan bobot-bobot yang membentuk barisan superincreasing adalah {2, 3, 6, 13, 27, 52}, dan diketahui bobot knapsack (M) = 70. Kita akan mencari b_1, b_2, …, b_6 sedemikian sehingga

$$70 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_6$$

Caranya sebagai berikut:

(i) Bandingkan 70 dengan bobot terbesar, yaitu 52. Karena 52 ≤ 70, maka 52 dimasukkan ke dalam knapsack.

(ii) Bobot total sekarang menjadi 70 – 52 = 18. Bandingkan 18 dengan bobot terbesar kedua, yaitu 27. Karena 27 > 18, maka 27 tidak dimasukkan ke dalam knapsack.

(iii) Bandingkan 18 dengan bobot terbesar berikutnya, yaitu 13. Karena 13 ≤ 18, maka 13 dimasukkan ke dalam knapsack.

(iv) Bobot total sekarang menjadi 18 – 13 = 5.

(v) Bandingkan 5 dengan bobot terbesar kedua, yaitu 6. Karena 6 > 5, maka 6 tidak dimasukkan ke dalam knapsack.

(vi) Bandingkan 5 dengan bobot terbesar berikutnya, yaitu 3. Karena 3 ≤ 5, maka 3 dimasukkan ke dalam knapsack.

(vii) Bobot total sekarang menjadi 5 – 3 = 2.

(viii) Bandingkan 2 dengan bobot terbesar berikutnya, yaitu 2. Karena 2 ≤ 2, maka 2 dimasukkan ke dalam knapsack.

(ix) Bobot total sekarang menjadi 2 – 2 = 0.

Karena bobot total tersisa = 0, maka solusi persoalan superincreasing knapsack ditemukan.
Barisan bobot yang dimasukkan ke dalam knapsack adalah

$\{2, 3, -, 13, -, 52\}$

Sehingga

$$70 = (1 \times 2) + (1 \times 3) + (0 \times 6) + (1 \times 13) + (0 \times 27) + (1 \times 52)$$

Dengan kata lain, plainteks dari kriptogram 70 adalah 110101.

13.4. Algoritma Knapsack Kunci Publik

Algoritma superincreasing knapsack adalah algoritma yang lemah, karena cipherteks dapat didekripsi menjadi plainteksnya secara mudah dalam waktu lanjar ($O(n)$).

Algoritma non-superincreasing knapsack atau normal knapsack adalah kelompok algoritma knapsack yang sulit (dari segi komputasi) karena membutuhkan waktu dalam orde eksponensial untuk memecahkannya.

Namun, superincreasing knapsack dapat dimodifikasi menjadi non-superincreasing knapsack dengan menggunakan kunci publik (untuk enkripsi) dan kunci privat (untuk dekripsi). Kunci publik merupakan barisan non superincreasing sedangkan kunci privat tetap merupakan barisan superincreasing. Modifikasi ini ditemukan oleh Martin Hellman dan Ralph Merkle. Cara membuat kunci publik dan kunci privat:

1. Tentukan barisan superincreasing.
2. Kalikan setiap elemen di dalam barisan tersebut dengan n modulo m. Modulus m seharusnya angka yang lebih besar daripada jumlah semua elemen di dalam barisan, sedangkan pengali n seharusnya tidak mempunyai faktor persekutuan dengan m.

3. Hasil perkalian akan menjadi kunci publik sedangkan barisan superincreasing semula menjadi kunci privat.

Contoh 13.3. Misalkan barisan superincreasing adalah {2, 3, 6, 13, 27, 52), m = 105, dan n = 31.

Barisan non-superincreasing (atau normal) knapsack dihitung sbb:

$2 \cdot 31 \bmod 105 = 62$
$3 \cdot 31 \bmod 105 = 93$
$6 \cdot 31 \bmod 105 = 81$
$13 \cdot 31 \bmod 105 = 88$
$27 \cdot 31 \bmod 105 = 102$
$52 \cdot 31 \bmod 105 = 37$

Jadi, kunci publik adalah {62, 93, 81, 88, 102, 37}, sedangkan kunci privat adalah {2, 3, 6, 13, 27, 52}.

Enkripsi

- Enkripsi dilakukan dengan cara yang sama seperti algoritma knapsack sebelumnya.
- Mula-mula plainteks dipecah menjadi blok bit yang panjangnya sama dengan kardinalitas barisan kunci publik.
- Kalikan setiap bit di dalam blok dengan elemen yang berkoresponden di dalam kunci publik.

Contoh 13.4. Misalkan

Plainteks: 0110001101011011 10

dan kunci publik yang digunakan seperti pada Contoh 3. Plainteks dibagi menjadi blok yang panjangnya 6, kemudian setiap bit di dalam blok dikalikan dengan elemen yang berkorepsonden di dalam kunci publik:

Blok plainteks ke-1 : 011000
Kunci publik : 62, 93, 81, 88, 102, 37
Kriptogram : $(1 \times 93) + (1 \times 81) = 174$

Blok plainteks ke-2 : 110101
Kunci publik : 62, 93, 81, 88, 102, 37
Kriptogram : $(1 \times 62) + (1 \times 93) + (1 \times 88) + (1 \times 37) = 280$

Blok plainteks ke-3 : 101110
Kunci publik : 62, 93, 81, 88, 102, 37
Kriptogram : $(1 \times 62) + (1 \times 81) + (1 \times 88) + (1 \times 102) = 333$

Jadi, cipherteks yang dihasilkan : 174, 280, 333

Dekripsi

- Dekripsi dilakukan dengan menggunakan kunci privat.
- Mula-mula penerima pesan menghitung n^{-1} , yaitu balikan n modulo m, sedemikian sehingga $n \cdot n^{-1} \equiv 1 \pmod{m}$. Kekongruenan ini dapat dihitung dengan cara yang sederhana sebagai berikut (disamping dengan cara yang sudah pernah diberikan pada Teori Bilangan Bulat):

$$n \cdot n^{-1} \equiv 1 \pmod{m}$$
$$\Leftrightarrow n \cdot n^{-1} = 1 + km$$
$$\Leftrightarrow n^{-1} = (1 + km)/n \quad , k \text{ sembarang bilangan bulat}$$

- Kalikan setiap kriptogram dengan n^{-1} mod m, lalu nyatakan hasil kalinya sebagai penjumlahan elemen-elemen kunci privat untuk memperoleh plainteks dengan menggunakan algoritma pencarian solusi superincreasing knapsack.

Contoh 13.5. Kita akan mendekripsikan cipherteks dari Contoh 13.4 dengan menggunakan kunci privat {2, 3, 6, 13, 27, 52}. Di sini, $n = 31$ dan $m = 105$. Nilai n^{-1} diperoleh sbb:

$$n^{-1} = (1 + 105k)/31$$

Dengan mencoba $k = 0, 1, 2, \ldots$, maka untuk $k = 18$ diperoleh n^{-1} bilangan bulat, yaitu

$$n^{-1} = (1 + 105 \cdot 18)/31 = 61$$

Cipherteks dari Contoh 16.4 adalah 174, 280, 222. Plainteks yang berkoresponden diperoleh kembali sebagai berikut:

$174 \cdot 61 \bmod 105 = 9 = 3 + 6,$
berkoresponden dengan 011000

$280 \cdot 61 \bmod 105 = 70 = 2 + 3 + 13 + 52,$
berkoresponden dengan 011000

$333 \cdot 61 \bmod 105 = 48 = 2 + 6 + 13 + 27,$
berkoresponden dengan 101110

Jadi, plainteks yang dihasilkan kembali adalah:
011000 011000 101110

13.5. Implementasi Knapsack

- Ukuran cipherteks yang dihasilkan lebih besar daripada plainteksnya, karena enkripsi dapat menghasilkan kriptogram yang nilai desimalnya lebih besar daripada nilai desimal blok plainteks yang dienkripsikan.
- Untuk menambah kekuatan algoritma knapsack, kunci publik maupun kunci privat seharusnya paling sedikit

250 elemen, nilai setiap elemen antara 200 sampai 400 bit panjangnya, nilai modulus antara 100 sampai 200 bit.

- Dengan nilai-nilai knapsack sepanjang itu, dibutuhkan 1046 tahun untuk menemukan kunci secara brute force, dengan asumsi satu juta percobaan setiap detik.
- Sayangnya, algoritma knapsack dinyatakan sudah tidak aman, karena knapsack dapat dipecahkan oleh pasangan kriptografer Shamir dan Zippel. Mereka merumuskan transformasi yang memungkinkan mereka merekonstruksi superincreasing knapsack dari normal knapsack.

14.
Fungsi Hash Satu-Arah dan Algoritma MD5

14.1. Pendahuluan

Fungsi hash adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi string keluaran yang panjangnya tetap (fixed) (umumnya berukuran jauh lebih kecil daripada ukuran string semula).

Fungsi hash dapat menerima masukan string apa saja. Jika string menyatakan pesan (message), maka sembarang pesan M berukuran bebas dikompresi oleh fungsi hash H melalui persamaan

$$h = H(M) \qquad \textbf{(14.1)}$$

Keluaran fungsi hash disebut juga **nilai hash** (*hash-value*) atau **pesan-ringkas** (*message digest*). Pada persamaan (14.1), h adalah nilai hash atau message digest dari fungsi H untuk masukan M.

Dengan kata lain, fungsi hash mengkompresi sembarang pesan yang berukuran berapa saja menjadi message digest yang ukurannya selalu tetap (dan lebih pendek dari panjang pesan semula).

Nama lain fungsi hash adalah:

- fungsi kompresi/kontraksi (compression function)
- cetak-jari (fingerprint)

- *cryptographic checksum*
- *message integrity check* (MIC)
- *manipulation detection code* (MDC)

Aplikasi fungsi hash misalnya untuk memverifikasi kesamaan salinan suatu arsip di dengan arsip aslinya yang tersimpan di dalam sebuah basis data terpusat.

Ketimbang mengirim salinan arsip tersebut secara keseluruhan ke komputer pusat (yang membutuhkan waktu transmisi lama), lebih mangkus mengirimkan message digest-nya. Jika message digest salinan arsip sama dengan message digest arsip asli, berarti salinan arsip tesrebut sama dengan asrip di dalam basis data.

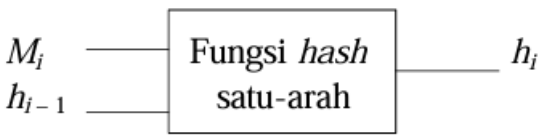
14.2. Fungsi Hash Satu-Arah (*One-way Hash*)

Fungsi hash satu-arah adalah fungsi hash yang bekerja dalam satu arah: pesan yang sudah diubah menjadi message digest tidak dapat dikembalikan lagi menjadi pesan semula
Sifat-sifat fungsi hash satu-arah adalah sebagai berikut:

1. Fungsi H dapat diterapkan pada blok data berukuran berapa saja.
2. H menghasilkan nilai (h) dengan panjang tetap (*fixed length output*).
3. $H(x)$ mudah dihitung untuk setiap nilai x yang diberikan.
4. Untuk setiap h yang dihasilkan, tidak mungkin dikembalikan nilai x sedemikian sehingga $H(x) = h$. Itulah sebabnya fungsi H dikatakan fungsi hash satu-arah (one way hash function).
5. Untuk setiap x yang diberikan, tidak mungkin mencari $y \neq x$ sedemikian sehingga $H(y) = H(x)$.
6. Tidak mungkin mencari pasangan x dan y sedemikian sehingga $H(x) = H(y)$.

Masukan fungsi hash adalah blok pesan (*M*) dan keluaran dari hashing blok pesan sebelumnya,

Skema fungsi hash ditunjukkan pada Gambar 14.1.



Gambar 14.1 Fungsi hash satu-arah

Fungsi hash adalah publik (tidak dirahasiakan), dan keamanannya terletak pada sifat satu arahnya itu.

Ada beberapa fungsi hash satu-arah yang sudah dibuat orang, antara lain:

- MD2, MD4, MD5,
- *Secure Hash Function* (SHA),
- *Snefru,*
- *N-hash*,
- *RIPE-MD*,
- dan lain-lain

(Catatan: MD adalah singkatan dari *Message Digest*).

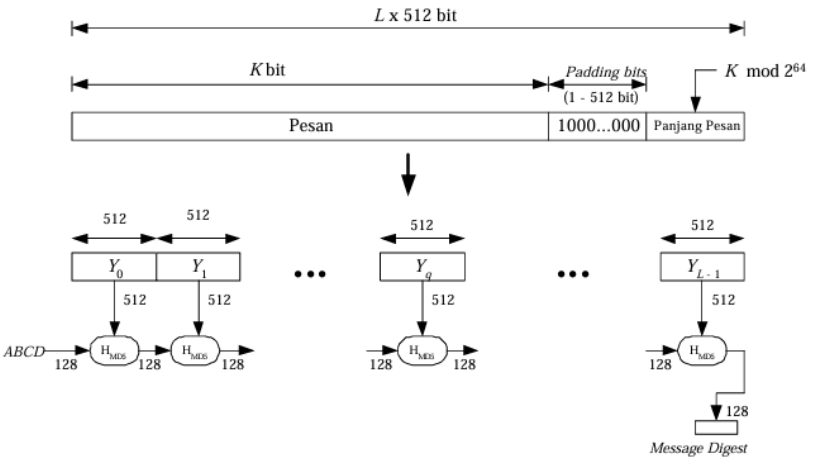
Fungsi *hash* yang banyak dipakai di dalam aplikasi kriptografi adalah *MD*5 dan *SHA*. Algoritma *MD*5 diberikan di bawah ini, sedangka SHA akan diberikan pada materi *DSS* (*Digital Signature Standard*).

14.3. Algoritma MD5

MD5 adalah fungsi hash satu-arah yang dibuat oleh Ron Rivest. MD5 merupakan perbaikan dari MD4 setelah MD4 berhasil diserang oleh kriptanalis. Algoritma MD5 menerima

masukan berupa pesan dengan ukuran sembarang dan menghasilkan message digest yang panjangnya 128 bit.

Gambaran pembuatan message digest dengan algoritma MD5 diperlihatkan pada Gambar 14.2.



Gambar 14.2 Pembuatan *message digest* dengan algoritma MD5

Langkah-langkah pembuatan message digest secara garis besar adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (padding bits).
2. Penambahan nilai panjang pesan semula.
3. Inisialisasi penyangga (buffer) MD.
4. Pengolahan pesan dalam blok berukuran 512 bit.

1. *Penambahan Bit-bit Pengganjal*

Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Ini berarti panjang pesan setelah ditambahi bit-bit pengganjal adalah 64 bit kurang dari kelipatan 512. Angka 512 ini muncul karena MD5 memperoses pesan dalam blok-blok yang berukuran 512.

ganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 sampai 512.

Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

2. ***Penambahan Nilai Panjang Pesan Semula***

Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan > 2^{64} maka yang diambil adalah panjangnya dalam modulo 2^{64}. Dengan kata lain, jika panjang pesan semula adalah *K* bit, maka 64 bit yang ditambahkan menyatakan *K* modulo 2^{64}.

Setelah ditambah dengan 64 bit, panjang pesan sekarang menjadi 512 bit.

3. ***Inisialisai Penyangga MD***

MD5 membutuhkan 4 buah penyangga (*buffer*) yang masing masing panjangnya 32 bit. Total panjang penyangga adalah 4 × 32 = 128 bit. Keempat penyangga ini menampung hasil antara dan hasil akhir.

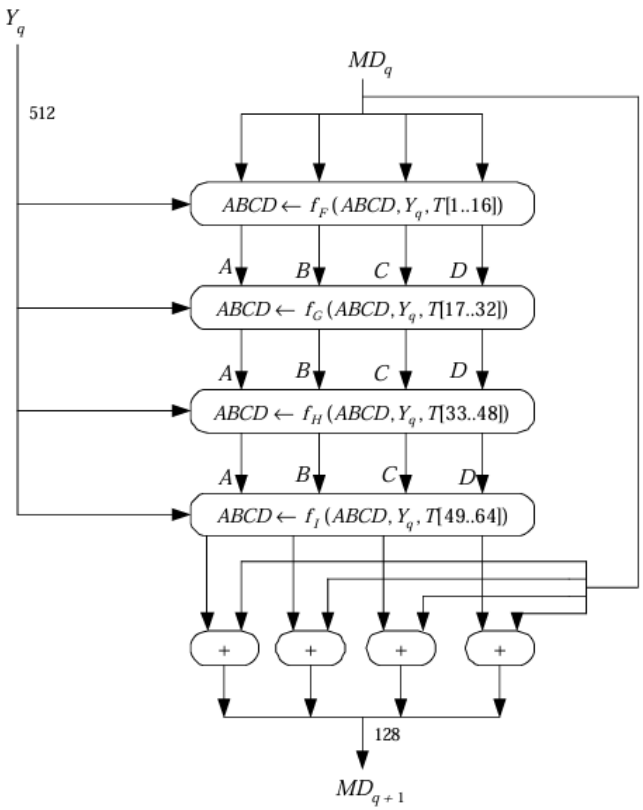
Keempat penyangga ini diberi nama *A, B, C*, dan *D*. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut:

A = 01234567
B = 89ABCDEF
C = FEDCBA98
D = 76543210

4. ***Pengolahan Pesan dalam Blok Berukuran 512 bit.***

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai $Y_L - 1$).

Setiap blok 512-bit diproses bersama dengan penyangga *MD* menjadi keluaran 128-bit, dan ini disebut proses H_{MD5}. Gambaran proses H_{MD5} diperlihatkan pada Gambar 14.3.



Gambar 14.3 Pengolahan blok 512 bit (Proses H_{MD5})

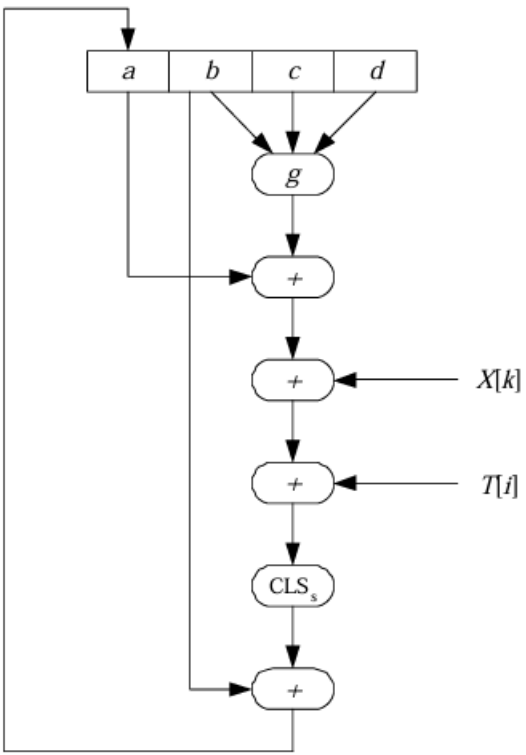
Proses H_{MD5} terdiri dari 4 buah putaran, dan masing-masing putaran melakukan operasi dasar *MD5* sebanyak 16 kali dan setiap operasi dasar memakai sebuah elemen *T*. Jadi setiap putaran memakai 16 elemen Tabel *T*.

Pada Gambar 17.3, Y_q menyatakan blok 512-bit ke-*q* dari pesan yang telah ditambah bit-bit pengganjal dan tambahan 64 bit nilai panjang pesan semula. MD_q adalah nilai message digest 128-

bit dari proses HMD5 ke-q. Pada awal proses, MDq berisi nilai inisialisasi penyangga MD. Proses HMD5 terdiri dari 4 buah putaran, dan masing-masing putaran melakukan operasi dasar MD5 sebanyak 16 kali dan setiap operasi dasar memakai sebuah elemen T. Jadi setiap putaran memakai 16 elemen Tabel T.

Pada Gambar 14.3, Y_q menyatakan blok 512-bit ke-q dari pesan yang telah ditambah bit-bit pengganjal dan tambahan 64 bit nilai panjang pesan semula. MD_q adalah nilai message digest 128-bit dari proses H_{MD5} ke-q. Pada awal proses, MD_q berisi nilai inisialisasi penyangga *MD*.

Fungsi-fungsi f_F, f_G, f_H, dan f_I masing-masing berisi 16 kali operasi dasar terhadap masukan, setiap operasi dasar menggunakan elemen Tabel *T*. Operasi dasar *MD5* diperlihatkan pada Gambar 14.4.



Gambar 14.4 Operasi dasar MD5

Operasi dasar *MD5* yang diperlihatkan pada Gambar 14.4 dapat ditulis dengan sebuah persamaan sebagai berikut:

$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i]) \quad \textbf{(14.7)}$$

yang dalam hal ini,

a, b, c, d = empat buah peubah penyangga 32-bit (berisi nilai penyangga A, B, C, D)

g = salah satu fungsi F, G, H, I

CLS_s = *circular left shift* sebanyak s bit

$X[k]$ = kelompok 32-bit ke-k dari blok 512 bit *message* ke-q. Nilai $k = 0$ sampai 15.

$T[i]$ = elemen Tabel T ke-i (32 bit)

$+$ = operasi penjumlahan modulo 2^{32}

Fungsi f_F, f_G, f_H, dan f_I adalah fungsi untuk memanipulasi masukan a, b, c, dan d dengan ukuran 32-bit. Masing-masing fungsi dapat dilihat pada Tabel 1.

Tabel 14.1 Fungsi-fungsi dasar MD5

Nama	Notasi	$g(b, c, d)$
f_F	$F(b, c, d)$	$(b \wedge c) \vee (\sim b \wedge d)$
f_G	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \sim d)$
f_H	$H(b, c, d)$	$b \oplus c \oplus d$
f_I	$I(b, c, d)$	$c \oplus (b \wedge \sim d)$

Catatan: operator logika AND, OR, NOT, XOR masing-masing dilambangkan dengan $\wedge$, $\vee$, $\sim$, $\oplus$

Nilai T[*i*] dapat dilihat pada Tabel 2. Tabel ini disusun oleh fungsi 2^{32} x abs(sin(*i*)), *i* dalam radian.

Tabel 14.2 Nilai T[*i*]

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 69D96122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBCB	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

Dari persamaan (7) dapat dilihat bahwa masing-masing fungsi fF, fG, fH, dan fI melakukan 16 kali operasi dasar.
Misalkan notasi

$$[abcd \quad k \quad s \quad i]$$

menyatakan operasi

$$a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i])<<<s)$$

yang dalam hal ini <<<*s* melambangkan operasi circular left shift 32-bit, maka operasi dasar pada masing-masing putaran dapat ditabulasikan sebagai berikut:

Putaran 1 : 16 kali operasi dasar dengan $g(b, c, d) = F(b, c, d)$ dapat dilihat pada Tabel 3.

Tabel 14.3 Rincian operasi pada fungsi *F*(*b, c, d*)

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	0	7	1]
2	[DABC	1	12	2]
3	[CDAB	2	17	3]
4	[BCDA	3	22	4]
5	[ABCD	4	7	5]
6	[DABC	5	12	6]
7	[CDAB	6	17	7]
8	[BCDA	7	22	8]
9	[ABCD	8	7	9]
10	[DABC	9	12	10]
11	[CDAB	10	17	11]
12	[BCDA	11	22	12]
13	[ABCD	12	7	13]
14	[DABC	13	12	14]
15	[CDAB	14	17	15]
16	[BCDA	15	22	16]

Putaran 2 : 16 kali operasi dasar dengan g(*b, c, d*) = *G*(*b, c, d*) dapat dilihat pada Tabel 4.

Tabel 14.4 Rincian operasi pada fungsi *G*(*b, c, d*)

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	1	5	17]
2	[DABC	6	9	18]
3	[CDAB	11	14	19]
4	[BCDA	0	20	20]
5	[ABCD	5	5	21]
6	[DABC	10	9	22]
7	[CDAB	15	14	23]
8	[BCDA	4	20	24]
9	[ABCD	9	5	25]
10	[DABC	14	9	26]
11	[CDAB	3	14	27]
12	[BCDA	8	20	28]
13	[ABCD	13	5	29]
14	[DABC	2	9	30]
15	[CDAB	7	14	31]
16	[BCDA	12	20	32]

Putaran 3 : 16 kali operasi dasar dengan $g(b, c, d) = H(b, c, d)$ dapat dilihat pada Tabel 5.

Tabel 14.5 Rincian operasi pada fungsi $H(b, c, d)$

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	5	4	33]
2	[DABC	8	11	34]
3	[CDAB	11	16	35]
4	[BCDA	14	23	36]
5	[ABCD	1	4	37]
6	[DABC	4	11	38]
7	[CDAB	7	16	39]
8	[BCDA	10	23	40]
9	[ABCD	13	4	41]
10	[DABC	0	11	42]
11	[CDAB	3	16	43]
12	[BCDA	6	23	44]
13	[ABCD	9	4	45]
14	[DABC	12	11	46]
15	[CDAB	15	16	47]
16	[BCDA	2	23	48]

Putaran 4 : 16 kali operasi dasar dengan $g(b, c, d) = I(b, c, d)$ dapat dilihat pada Tabel 6.

Tabel 14.6 Rincian operasi pada fungsi *I*(*b, c, d*)

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	0	6	49]
2	[DABC	7	10	50]
3	[CDAB	14	15	51]
4	[BCDA	5	21	52]
5	[ABCD	12	6	53]
6	[DABC	3	10	54]
7	[CDAB	10	15	55]
8	[BCDA	1	21	56]
9	[ABCD	8	6	57]
10	[DABC	15	10	58]
11	[CDAB	6	15	59]
12	[BCDA	13	21	60]
13	[ABCD	4	6	61]
14	[DABC	11	10	62]
15	[CDAB	2	15	63]
16	[BCDA	9	21	64]

Setelah putaran keempat, *a, b, c,* dan *d* ditambahkan ke *A, B, C,* dan *D*, dan selanjutnya algoritma memproses untuk blok data berikutnya (Y_{q+1}). Keluaran akhir dari algoritma *MD5* adalah hasil penyambungan bit-bit di *A, B, C*, dan *D*.

Dari uraian di atas, secara umum fungsi hash MD5 dapat ditulis dalam persamaan matematis berikut:

$$MD_0 = IV \qquad \textbf{(14.8)}$$

$$MD_{q+1} = MD_q + f_I(Y_q + f_H(Y_q + f_G(Y_Q + f_F(Y_q + MD_q))))$$

$$MD = MD_{L-1} \qquad \textbf{(14.8)}$$

yang dalam hal ini,

IV = *initial vector* dari penyangga ABCD, yang dilakukan pada proses inisialisasi penyangga.
Y_q = blok pesan berukuran 512-bit ke-q
L = jumlah blok pesan
MD = nilai akhir *message digest*

Contoh 14.1 Misalkan M adalah isi sebuah arsip teks bandung.txt sebagai berikut:

```
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 33 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.
```

Message digest dari arsip bandung.txt yang dihasilkan oleh algoritma MD5 adalah 128-bit:

```
0010 1111 1000 0010 1100 0000 1100 1000 1000
0100 0101 0001 0010 0001 1011 0001 1011 1001
0101 0011 1101 0101 0111 1101 0100 1100 0101
1001 0001 1110 0110 0011
```

atau, dalam notasi HEX adalah:

```
2F82D0C845121B953D57E4C3C5E91E63
```

14.4. Aplikasi Fungsi Hash untuk Integritas Data

Kadang-kadang kita menginginkan isi arsip tetap terjaga keasliannya (tidak diubah oleh orang yang tidak berhak). Perubahan kecil pada isi arsip sering tidak terdeteksi, khususnya pada asrip yang berukuran besar.

Fungsi hash dapat digunakan untuk menjaga keutuhan (integritas) data. Caranya, bangkitkan message digest dari isi arsip (misalnya dengan menggunakan algoritma MD5). Message digest dapat digabung ke dalam arsip atau disimpan di dalam arsip.

Verifikasi isi arsip dapat dilakukan secara berkala dengan membandingkan message digest dari isi arsip sekarang dengan message digest dari arsip asli. Jika terjadi perbedaan, maka disimpulkan ada modifikasi terhadap isi arsip (atau terhadap message digest yang disimpan).

Aplikasi ini didasarkan pada kenyataan bahwa perubahan 1 bit pada pesan akan mengubah, secara rata-rata, setengah dari bit-bit messsage digest. Dengan kata lain, fungsi hash sangat peka terhadap perubahan sekecil apa pun pada data masukan.

Contoh 17.2 Misalkan message digest dari arsip bandung.txt disertakan di dalam arsip bersangkutan sebagai baris pertama:

```
2F82D0C845121B953D57E4C3C5E91E63
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 33 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.
```

Kasus 1: Misalkan 33 derajat celcius diubah menjadi 32. Message digest dari isi arsip (tidak termasuk baris message digest) adalah:

Sebelum diubah : MD1 = 2F82D0C845121B953D57E4C3C5E91E63
Sesudah diubah : MD2 = 2D1436293FAEAF405C27A151C0491267
Verifikasi: MD1 ≠ MD2 (arsip sudah diubah)

Kasus 2: Ditambahankan sebuah spasi antara "33" dengan kata "derajat". Message digest dari isi arsip (tidak termasuk baris message digest) adalah:

Sebelum diubah : MD1 = 2F82D0C845121B953D57E4C3C5E91E63
Sesudah diubah : MD2 = F8F6AB94724E584277D77b4185CF21DD
Verifikasi: MD1 ≠ MD2 (arsip sudah diubah)

Pengubahan juga dapat dilakukan terhadap message digest.

PENUTUP

Melalui pembelajaran buku ini, diharapkan akan membantu mahasiswa akan dapat belajar secara mandiri, mengukur kemampuan diri sendiri, dan menilai dirinya sendiri. Tidak terkecuali dalam memahami Metode Penelitian. Semoga buku ini dapat digunakan sebagai referensi tambahan dalam proses pembelajaran Kriptografi dan Sekuriti Sistem.

Pembaca dimohon dapat lebih mendalami materi lain di samping materi yang ada di buku ini melalui berbagai sumber, jurnal, maupun internet. Semoga buku ini bermanfaat bagi pembaca. Tak lupa dalam kesempatan ini, penulis mohon saran dan kritik yang membangun terhadap isi buku ini, demi sempurnanya penyusunan buku ini di masa-masa yang akan datang. Semoga buku ini memberikan manfaat bagi pembaca budiman lainnya.

DAFTAR **PUSTAKA**

Alabaichi. A, "True Color Image Encryption Based On Dna Sequence, 3d Chaotic Map, And Key-Dependent Dna S-Box Of Aes," *Journal of Theoretical & Applied Information Technology,* vol. 96, no. 2, pp. 304-321, 2018.

Adilah, S & Paryasto, W. (2017) Implementasi Kriptosystem menggunakan metode Algoritma ECC dengan Fungsi Hash SHA-256 pada sistem ticketing online, *e-Proceeding of Engineering,* p. 4140.

Aldyza, A. (2016) Analisis Perbandingan Keamanan Kriptografi Klasik Pada Algoritma Modifikasi Playfair Cipher berbasis Matriks 8x8 dan 16x16," *Jurnal Ilmu Komputer dan Informatika*

Apridiansyah, Y & Rifqo, M. (2015) Aplikasi Keamanan Lembar Hasil Studi Menggunakan Algoritma *Message Digest* 5, *Jurnal Pseudocode,* p. 108.

Arab. A, Rostami, M. J dan Ghavami, B. "An image encryption method based on chaos system and AES algorithm," *The Journal of Supercomputing,* vol. 75, no. 10, pp. 6663-6682, 2019.

Bhat, v. & Shridhar (2012). "FPGA Implementations of Advanced Encryption Standard: a survey," International Journal of Advances In Engineering & Technology, vol. 3, issue 2, pp. 265-285,

Choudhury, Prakash, K & Kakoty, S. (2017) *Comparative Analysis of Different Modified AES Algorithms Over Conventional Algotihms*, *International Journal of Current Research and Review,* vol. 9, no. 22

Dawood, O & Hammadi, I. (2017) *An Analytical Study for Some Drawbacks and Weakness Points of the AES Cipher (Rijndael Algorithm),The 1st International Conference on Information Technology*

H. V. Gamido, A. M. Sison dan R. P. Medina, "Implementation of modified AES as image encryption scheme," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI),* vol. 6, no. 3, pp. 301-308, 2018.

Haris, Muhammad (2021) "Pengamanan Pada Citra Digital dengan Menggunakan Modifikasi Blok Data Algoritma AES-Rijndael" Jurnal Media Informatika Budidarma 7 (1), 444-453, 2023.

I. N. Ibraheem, S. M. Hassan dan A. Abead, "Comparative Analysis & Implementation of Image Encryption & Decryption for Mobile Cloud Security," *International Journal of Advanced Science and Technology,* vol. 29, no. 3s, pp. 109-121, 2020

J. Hao, H. Li, H. Yan dan J. Mou, "A New Fractional Chaotic System and Its Application in Image Encryption With DNA Mutation," *IEEE Access,* vol. 9, pp. 52364-52377, 2021.

Mahajan, P & Sachdeva, A. (2013) *A Study of Encryption Algorithms* AES, DES and RSA for. *Global Journal of Computer Science and Technology,* vol. Volume 13 Issue 15 Version 1.0 no. ISSN: 0975-4172 & Print ISSN: 0975-4350

Mahalakshmi, J. (2012) *A New Technique for Image Encrpytion using Rijndael Block Cipher Algorithm, Internasional Journal of Computer Applications (0975-8887)* vol. 45, No 11

Mathur, B. (2017) *Accountability of Civil Service – Creating An Institutional Framework, India Journal of Public Administrational*

Mukhtar, H. (2018) *kriptografi Untuk Keamanan Data/oleh Harun Mukhtar,--Ed.1, Cet 1-- Yogyakarta: Deepublish, November* ISBN : 978-602-475-946-9

Muttaqin, K & Rahmadoni, J. (2020) *Analysis And Design of File Security System AES (Advance Encryption Standard) Cryptography Based,*" *Journal of Applied Engineering and Technological Science,* vol. 1 no. 2.

N. Nguyen-Thanh, D. Marinca, K. Khawam, S. Martin dan L. Boukhatem, "Multimedia content popularity: Learning and recommending a prediction method," dalam *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018.

Nisak. K. (2015). Penyandian Kriptografi Metode Hill Cipher Dan Caesar Cipher Dengan Menggunakan Appinventor. Skripsi Jurusan Matematika Fakultas Sains Dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

O. F. Mohammad, M. S. M. Rahim, S. R. M. Zeebaree dan F. Y. Ahmed, "A survey and analysis of the image encryption methods," *International Journal of Applied Engineering Research,* vol. 12, no. 23, pp. 13265-13280, 2017.

P. Sharma dan H. Sabharwal, "A New Image Encryption using Modified AES Algorithm and its Comparision with AES," *International Journal Of Engineering Research & Technology (IJERT),* vol. 9, no. 8, pp. 194-197, 2020.

Putri, G , Styorini, W & Rahayani, D. (2018) Analisis Kriptografi Simetris AES Dan Kriptografi Asimetris RSA Pada Enkripsi Citra Digital, *Ethos (Jurnal Penelitian dan Pengabdian Masyarakat),* p. 198

Riza, F. (2018). Analisa Frekuensi Hasil Enkripsi Algoritma Blowfish Terhadap Keamanan Informasi," *Publikasi Jurnal Penelitian Teknik Informatika,* vol. 1, pp. 11-15.

Shulhan, I . (2018) Analisis Perbandingan Antara Algoritma Rijndael Dan Algoritma Twofish Dalam Penyandian Teks. *Jurnal Teknik Informatika Unika St. Thomas (JTIUST),* Volume 03 Nomor 02, Desember 2018, ISSN: 2548-1916

Siambaton, Zulfansyuri, M & Muhazzir, A. (2018) . Modifikasi Algoritma *Playfair Cipher* Dengan Pengurutan *Array* pada Matriksv. *Jurnal Ilmu Komputer dan Informatika,* vol. 02, no. ISSN 2598-6341.

Siddaiah, N, Moduli, A, Bhargauram, K . (2020). *A Novel Approch to Modified Advance Encryption Standart Algorithm*. *International Jornal of Engineering and Advanced Technology*. Vol 7, Issue 2, ISSN.2394-5125

Sianipar, K , Sihaan, W , Siregar, M , & Gunawan, I (2019) Pengamanan File Suara Menggunakan Kriptografi Algortima Rijndael Dengan Prosse Enkripsi dan Dekripsi. TECHSI: Vol. 11, No. 3

Soumya, K & Kishore, G. (2013) *Desain and Implementation of Rijndael Encryption Algorithm Based on FPGA*, *International Conference on Computer and Electrical Engineering, Corpus ID :* 212493631

Susanti, D. (2020). Analisis Modifikasi Metode Playfair Cipher Dalam Pengamanan Data Teks. *Indonesian Journal of Data and Science,* Vol 1, No 1, pp. 11-18, no. ISSN: 2715-993

Surian, D. (2006). Algoritma Kriptografi AES Rijndael. *Jurnal Teknik Elektro (TESLA),* Vol. 8, No. 2, 97 - 101

Stallings, William. (2005). Cryptography and Network Security Principles and Practices, Fourth Edition. Prentice Hall.

S. T. Kamal, K. M. Hosny, T. M. Elgindy, M. M. Darwish dan M. M. Fouda, "A new image encryption algorithm for grey and color medical images," *IEEE Access,* vol. 9, pp. 37855-37865, 2021.

Kristoforus, R & Aditya, S. (2012) Implementasi Algoritma Rijndael Untuk Enkripsi dan Dekripsi Pada Citra Digital.

Seminar Nasional Aplikasi Teknologi Informasi No. ISSN : 1907 – 5022.

Walton, J. (2008). Applied Crypto++: Block Ciphers. (Online), (http://www.codeproject.com/KB/security/BlockCiphers.aspx), diakses 21 September 2011.

Y. Pourasad, R. Ranjbarzadeh dan A. Mardani, "A New Algorithm for Digital Image Encryption Based on Chaos Theory," *Entropy,* vol. 23, no. 3, p. 341, 2021.

Yuniati, T, Suryani, E & Aziz, A. (2012). Pengaruh Variasi Panjang Kunci, Ukuran Blok, dan Mode Operasi Terhadap Waktu Eksekusi pada Algoritma Rijndael. *Jurnal ITSmart*. Vol 1. No 1. ISSN : 2301–7201

Zebua, T & Ndruru, E. (2017). Pengamana Citra Digital Berdasarkan Modifikasi Algoritma RC4. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK).* Vol. 4, No. 4, Desember 2017, hlm. 275-282

GLOSARIUM

Plainteks	Digunakan dalam konteks kriptografi untuk merujuk pada teks atau data yang belum dienkripsi. Dengan kata lain, plainteks adalah bentuk asli dari sebuah pesan atau informasi sebelum diterapkan proses enkripsi untuk mengamankannya.
Cipherteks	Hasil dari proses enkripsi pada sebuah teks atau data asli (plainteks). Cipherteks merupakan bentuk data yang telah diubah menggunakan algoritma enkripsi sehingga tidak dapat dibaca atau dipahami tanpa proses dekripsi menggunakan kunci yang sesuai.
Kriptografi	Ilmu dan seni mengamankan informasi dengan cara mengubahnya menjadi format yang tidak dapat dibaca oleh pihak yang tidak berwenang. Kriptografi digunakan untuk melindungi data selama penyimpanan maupun pengiriman, memastikan kerahasiaan, keaslian, integritas, dan keabsahan informasi.
Kriptanalis	Individu atau profesional yang mempelajari dan menganalisis

	sistem enkripsi dengan tujuan mengidentifikasi kelemahan, memecahkan cipherteks, atau mengungkapkan plainteks tanpa mengetahui kunci enkripsi.
Enkripsi	proses mengubah data atau informasi dalam bentuk asli (disebut plainteks) menjadi bentuk yang tidak dapat dibaca atau dimengerti (disebut cipherteks) menggunakan algoritma enkripsi dan kunci tertentu
Kriptologis	Merujuk pada bidang ilmu yang mempelajari semua aspek yang berkaitan dengan kriptografi dan kriptanalisis. Kriptologis mencakup teori, prinsip, teknik, dan aplikasi yang digunakan untuk mengamankan data dan komunikasi, serta menganalisis atau memecahkan sistem keamanan tersebut.
Watermark	Tanda atau simbol yang ditambahkan pada dokumen, gambar, video, atau file digital lainnya untuk menunjukkan kepemilikan, hak cipta, atau untuk tujuan keamanan. Watermark dapat berupa teks, logo, atau pola transparan yang biasanya tidak mengganggu konten utama tetapi tetap terlihat.

PROFIL **PENULIS**

Ferdy Riza, ST, M.Kom lahir di Kabupaten 50 Kota Sumatera Barat 3 Juni 1989. Beliau Menyelesaikan pendidikan S1 di Sekolah Tinggi Teknik Harapan (sekarang Universitas Harapan) pada tahun 2011 dan melanjutkan ke jenjang Magister ilmu Komputer di Universitas Putra Indonesia Padang yang selesai pada tahun 2013.

Muhammad Haris, S.Kom., M.Kom Muhammad Haris adalah salah satu dosen program studi teknologi informatika. Lahir di Medan 16 Agustus 1996, melakukan Pendidikan di bangku Sekolah Dasar di SD Swasta Muhammadiyah 28 Kota Medan, Kemudian melanjutkan SMP di SMP Islam Al – Ulum Terpadu Medan, Kemudian melanjutkan ke jenjang SMA di Madrasah Aliyah Negeri 3 Medan. Setelah Tamat dari Madrasah Aliyah Negeri 3 Medan, Beliau melanjutkan Pendidikan Tinggi Strata 1 di Universitas Harapan Medan Jurusan Teknik Informatika dan lulus pada tahun 2019.

Setelah lulus S1 kemudian meneruskan Pendidikan S2 di Universitas Sumatera Utara Jurusan Magister Komputer dan lulus pada tahun 2023. Pada saat S2 kerja di SD Muhammadiyah sebagai Pegawai Honor dan sebagai Operator Sekolah. Saat di S2 beliau melakukan penelitian dengan menulis jurnal tentang

kriptografi yang berjudul Pengaman Pada Citra Digital dengan Menggunakan Modifikasi Blok Data Algoritma AES – Rijndael dengan indeks jurnal sinta 3 di jurnal Media Informatika Budidarma Kota Medan. Penulis bisa dihubungi melalui 0878916860 dan surel muhammadharis@umsu.ac.id.

Mahardika Abdi Prawira Tanjung, S.Kom, M.Kom lahir di Medan 17 Agustus 1989. Pria yang telah selesai menempuh pendidikan sarjana di Universita Komputer Indonesia, dan Magister Teknik Informatika Universitas Sumatera Utara aktif mengajar di Program Studi Teknologi Informasi Universitas Muhammadiyah Sumatera Utara. Aktif melakukan sejumlah publikasi di bidang ilmu komputer. Kepedulian terhadap perkembangan teknologi informasi mendasari lahirnya buku yang berkolaborasi dengan mahasiswa ini. Publikasi yang dihasilkan diantaranya adalah "Analisis Pengaruh Storytelling Terhadap Game Lorong Waktu–Pangeran Dipenogoro Sebagai Media Edukasi Sejarah" (KOMPUTA, 2013), "*Modification of speed-up robust feature method with histogram of oriented gradient in image blur classification*" (ICTS, 2017), "Perancangan sistem layanan informasi hotel di lokasi wisata danau toba berbasis mobile" (SENAR, 2018), "Analisis Dan Perancangan Webbase Dan Wap Ticket Center Reservation Pada Po. Npm Dengan Metode Prosedur Multi User" (JURTEKSI, 2018), "Klasifikasi Kategori Citra Digital Dengan Metode *Bag Of Visual Words*" (JURTEKSI, 2019), "*Lamp Control System Through Android And Wifi Based On Arduino Microcontroller*" (JURTEKSI, 2020), ""Kian Santang" *game as historical educational media using digital storytelling concept*" (EAIT, 2020), "*Numerical Analysis of Variations Distance Formulas on K Nearest Neighbors In Classifying Malaria Parasite Blood Cells*" (JITE, 2022), "*The Comprehension of*

Tempo's Newspaper English Article for the 10th Grade Students at Al-Qomariyah Boarding School" (IJETH, 2022), "*Multivariate Analysis Approach to Factor-Affected Tuberculosis Disease*" (KEDS, 2022). Penulis bisa dihubungi melalui 082112719104 dan surel mahardikaabdiprawira@umsu.ac.id.

Farid Akbar Siregar, S.Kom., M.Kom lahir di Medan Sumatera Utara 4 April 1994. Beliau Menyelesaikan pendidikan S1 di Sekolah USU Ilmu Komputer dan selesai pada tahun 2016 dan melanjutkan ke jenjang Magister Teknik Informatika Universitas Sumatera Utara selesai pada tahun 2018.

Amrullah, S.Kom., M.Kom, Lahir di Medan Tahun 1986. Beliau menyelesaikan pendidikan dasar di SD Al-Washliyah 45 Medan tahun 1998 , Kemudian dilanjutkan SLTP Negeri 19 Medan tahun 2001 dan SMK TI Bina Satria Medan tahun 2004.
Wisuda Diploma 3 di STMIK Triguna Dharma Medan pada bidang ilmu Manajemen informatika pada tahun 2014, Wisuda Strata 1 di Triguna Dharma Medan pada Bidang Ilmu Sistem Informasi pada tahun 2016 dan menyelesaikan program Magister Komputer Fakultas Ilmu Komputer program studi Teknik Informatika konsentrasi Sistem Informasi pada UPI YPTK Padang tahun 2019.
Sebagai seorang dosen, Aktif menulis di berbagai jurnal nasional maupun jurnal internasional. Selain sebagai dosen pada FIKTI, beliau juga aktif sebagai author pada website internasional digital asset design dan typography font yang dapat di lihat

pada myfont.com Hasil karya beliau juga tersebar diberbagai marketplace nasional lainnya seperti Envato, Creative market.

Fatma Sari Hutagalung, S.Kom., M.Kom. Lahir dimedan tgl 17 januari 1993, Pendidikannya diawali dengan menjadi siswi di SDN 104081, Kemudian melanjutkan di SMPN9 Medan, dan dilanjutkan di SMAN15 Medan. Pada tahun 2016 melanjutkan pendidikan Strata 1 di Universitas Sumatera Utara pada bidang ilmu komputer. Dan melanjutkan pendidikan ke Strata-II bidang teknik informatika di universitas sumatera utara.

Mulkan Azhari, S.Kom, M.Kom, Saya seorang laki-laki berusia 25 tahun. Saya lahir di Medan pada tanggal 08 Desember 1994. Saya merupakan lulusan S1 dari Universitas Potensi Utama tahun 2018 dan telah lulus Strata 2 di Universitas Potensi Utama pada tahun 2021.

Zuli Agustina Gultom, M.SI, lahir di Batangtoru tangga; 30 Agustus 1990. Pendidikan sekolah dasar di SDN.100715 tahun 2001, Kemudian melanjutkan Sekolah menengah pertama di SMP NEGERI 1 Batangtoru dan SMA NEGERI 1 Sibolga. Wisuda D-III Statistika USU pada tahun 2011, Melanjutkan

program Sarjana Statistika ITS pada tahun 2014 dan Megister Statistika ITS pada tahun 2016.

Andi Zulherry, S.Kom., M.Kom Lahir di Galang Tahun 1979. Beliau menyelesaikan pendidikan dasar di SD Negeri 1 Perbaungan tahun 1992 , Kemudian dilanjutkan SMP Negeri 1 Perbungan tahun 1995 dan SMU Swasta Al-Azhar Medan tahun 1998. Lalu telah menyelesaikan Pendidikan Strata 1 di universitas Gunadarma di Depok, Jawa Barat pada tahun 2003 dan menyelesaikan Magister di Uiversitas Potensi Utama Medan, Sumatera Utara pada tahun 2021.

Hevlie Winda Nazry S, S.Pd, M.Si, Lahir di Medan pada tanggal 29 Juli Tahun 1993. Menyelesaikan pendidikan dasar di SD Persatuan Amal Bakti (PAB) 29 Helvetia tahun 2005 , Kemudian dilanjutkan SMP di sekolah Negeri 1 Labuhan Deli tahun 2008 dan SMA di Negeri 7 Medan tahun 20011. Wisuda Sarjana di Universitas Muhammadiyah Sumatera Utara (UMSU) Medan pada bidang ilmu pendidikan matematika pada tahun 2015, dan menyelesaikan program Magister di Fakultas Ilmu Matematika dan IPA pada program studi Matematika di Universitas Sumatera Utara (USU) tahun 2017.

Muhammad Basri, S.Si., M.Kom. Lahir di Binjai pada tanggal 11 Juli Tahun 1988. Menyelesaikan pendidikan dasar di SD Negeri 112162 Rantau Prapat (SDN 13 Rantau Utara) tahun 2000 , Kemudian dilanjutkan SMP di sekolah Negeri 1 Rantau Utara tahun 2003 dan SMA di Negeri 1 Rantau Utara tahun 2006. Wisuda Sarjana di Universitas Negeri Medan (UNIMED) Medan pada bidang ilmu matematika pada tahun 2011, dan menyelesaikan program Magister di Fakultas Ilmu Komputer dan Tekonologi Informasi pada program studi Teknik Informatika di Universitas Sumatera Utara (USU) tahun 2017.

Okvi Nugroho, S.Kom., M.Kom putra kelahiran 1993 di Medan, Indonesia. Telah menyelesaikan pendidikan akhir di Universitas Sumatera Utara (USU) pada Program Magister teknik Informatika pada tahun 2021. Pada awal perjalanan hidupnya aktif sebagai Engginer pada perusahaan swasta yang bergerak pada bidang telekomunikasi berbasis satelit. Namun seiring dengan perjalanan beliau aktif menjadi pengajar di Universitas Muhammadiyah Sumatera Utara (UMSU). Aktif pada bidang Artificial Intelligence, Machine learning , Natural language Processing, data mining dan data science. Aktif pada penulisan karya ilmiah yang dipublikasikan pada jurnal dan konferensi baik nasional maupun internasional.

EDITOR

Dr. Al-Khowarizmi, S.Kom., M.Kom. merupakan putra kelahiran tahun 1992 di Kota Medan yang telah menyelesaikan Pendidikan terakhirnya di Universitas Sumatera Utara (USU) pada Program Doktor Ilmu Komputer tahun 2023. Awal mula perjalanan hidup dia aktif menjadi praktisi IT di Provinsi Sumatera Utara dan telah menjadi tenaga ahli di beberapa Kabupaten/Kota. Namun, seiring dengan keputusannya untuk mengabdi pada Universitas Muhammadiyah Sumatera Utara (UMSU) dia aktif dalam menjalankan tri dharma dalam bidang pendidikan seperti mengampuh matakuliah kecerdasan buatan dan data mining, melakukan penelitian dalam bidang *Artificial Intelligence*, *data mining*, *Neural Network* dan *Machine Learning* yang dipublikasikan pada jurnal dan konferensi baik nasional dan international, serta melakukan pengabdian kepada masyarakat. Saat ini, sesuai dengan amanah yang diberikan, dia sedang menduduki jabatan sebagai Dekan Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) di UMSU periode 2021-2025. Selain itu, dia juga aktif dalam kegiatan asosiasi seperti menduduki jabatan Bendahara APTIKOM Sumatera Utara, Wakil ketua IPKIN Cabang Sumatera Utara, Wakil Sekretaris HIPMI Medan, Wakil Ketua LPPK Muhammadiyah Medan, Anggota Asosiasi Sains dan Teknologi Perguruan Tinggi Muhammadiyah (AST-PTM), Anggota Forum Dekan Teknik Indonesia (FDTI), dan lain sebagainya.

INDEKS

KRIPTOGRAFI
Dan Sekuriti Sistem

Buku Kriptografi dan Sekuriti Sistem disusun sebagai panduan bagi pembaca yang sedang mempelajari ilmu kriptografi dan sekuriti sistem. Fokus utama buku ini adalah memberikan pengantar yang jelas mengenai kriptografi, serta menjelaskan konsep dasar, peran utama, metode, proses, dan algoritma yang digunakan dalam kriptografi dan sekuriti sistem. Meskipun buku ini masih dalam tahap pengembangan dan belum sempurna, penyusun berharap agar pembaca dapat memberikan saran untuk perbaikan. Buku ini bertujuan untuk meningkatkan pemahaman dan keterampilan pembaca dalam menguasai topik kriptografi dan sekuriti sistem, serta memberikan kontribusi positif bagi perkembangan ilmu di bidang ini.

TEKNOLOGI

ISBN 978-623-408-895-3
9 786234 088953
ISBN 978-623-408-900-4 (PDF)
9 786234 089004
Harga P. Jawa Rp. 81.000,-

UMSU PRESS
Jalan Kapten Muktar Basri No. 3
Medan, Sumatera Utara
Website : http://umsupress.umsu.ac.id/
email : umsupress@umsu.ac.id