Secara garis besar, proses enkripsi adalah proses pengacakan "naskah asli" (*plaintext*) menjadi "naskah acak" (*ciphertext*) yang "sulit untuk dibaca" oleh seseorang yang tidak mempunyai kunci dekripsi. Yang dimaksud dengan "sulit untuk dibaca" disini adalah probabilitas mendapat kembali naskah asli oleh seseorang yang tidak mempunyai kunci dekripsi dalam waktu yang tidak terlalu lama adalah sangat kecil. Jadi suatu proses enkripsi yang baik menghasilkan naskah acak yang memerlukan waktu yang lama (contohnya satu juta tahun)[1] untuk didekripsi oleh seseorang yang tidak mempunyai kunci dekripsi. Satu cara untuk mendapatkan kembali naskah asli tentunya dengan menerka kunci dekripsi, jadi proses menerka kunci dekripsi harus menjadi sesuatu yang sulit. Tentunya naskah acak harus dapat didekripsi oleh seseorang yang mempunyai kunci dekripsi untuk mendapatkan kembali naskah asli.

Walaupun awalnya kriptografi digunakan untuk merahasiakan naskah teks, kini kriptografi digunakan untuk data apa saja yang berbentuk digital.

2.1 Konsep Acak

Yang dimaksud dengan sifat acak (*randomness*) dalam kriptografi adalah sifat bebas dari kecenderungan sehingga tidak mudah untuk diterka. Dari segi matematika, jika suatu variabel dianggap bersifat acak, maka teori probabilitas dapat digunakan untuk memprediksi "kelakuan" dari variabel tersebut, antara lain variabel akan memenuhi beberapa kriteria statistik. Metode statistika dapat digunakan, berdasarkan apa yang sudah terjadi, untuk menilai apakah variabel memenuhi kriteria statistik untuk variabel acak. Akan tetapi jika kriteria statistik terpenuhi, belum tentu variabel benar acak, karena sesuatu yang deterministik seperti *pseudo-random number generator* dapat memenuhi kriteria statistik untuk variabel acak. Jadi kriteria statistik bukan merupakan definisi untuk variabel acak.

Sifat acak memang tidak dapat didefinisikan secara matematis, sebab sesuatu yang mempunyai definisi matematis sudah tidak bersifat acak. Apalagi jika definisi berupa rumus yang dapat digunakan untuk kalkulasi, yang didefinisikan bukan saja mudah diterka tetapi tidak perlu diterka.

Sifat acak dapat dikaitkan dengan urutan *events*, dimana *event* berikutnya dalam suatu urutan tidak mudah untuk diterka berdasarkan apa yang sudah lalu. Sifat ini diperlukan dalam pembuatan kunci (*key generation*) supaya kunci dekripsi tidak mudah untuk diterka.

Sifat acak juga dikaitkan dengan tidak adanya korelasi (atau korelasi yang mendekati nol). Dalam kriptografi, tidak diinginkan adanya korelasi antara naskah asli dengan naskah acak atau kunci dengan naskah acak. Ini untuk

[1]Dalam kriptografi, waktu yang dimaksud adalah rerata waktu, ada kemungkinan waktu lebih singkat dan ada kemungkinan waktu lebih lama.

mempersulit analisa seperti analisa frekuensi (*frequency analysis*) atau analisa lebih canggih seperti *linear cryptanalysis* atau *differential cryptanalysis*.

Meskipun tidak sebenarnya acak, sesuatu yang *pseudo-random* berguna dan digunakan dalam kriptografi, tetapi harus dikombinasikan dengan sesuatu yang benar acak. Sebagai contoh, *pseudo-random number generator* dikombinasikan dengan sumber entropi yang benar acak sebagai *seed*, untuk mendapatkan sesuatu yang praktis bersifat *random number generator*.

2.2 One-Time Pad

Secara teoritis, teknik *one-time pad* merupakan teknik enkripsi yang sempurna (*perfect encryption*) asalkan proses pembuatan kunci benar acak.

10010111001011101001	naskah asli
01001110001101001101	kunci
11011001000110100100	naskah acak

Tabel 2.1: Proses enkripsi one-time pad

Dengan *one-time pad*, operasi *exclusive or* (xor) dilakukan pada naskah asli dan kunci secara *bitwise* seperti dalam tabel 2.1. Operasi xor menghasilkan 0 jika argumen sama (0 dengan 0 atau 1 dengan 1) dan menghasilkan 1 jika argumen berbeda (0 dengan 1 atau 1 dengan 0). Jadi bit pertama naskah asli (1) dengan bit pertama kunci (0) menghasilkan bit pertama naskah acak (1), bit kedua naskah asli (0) dengan bit kedua kunci (1) menghasilkan bit kedua naskah acak (1), bit ketiga naskah asli (0) dengan bit ketiga kunci (0) menghasilkan bit ketiga naskah acak (0), dan seterusnya.

11011001000110100100	naskah acak
01001110001101001101	kunci
10010111001011101001	naskah asli

Tabel 2.2: Proses dekripsi one-time pad

Proses dekripsi sama dengan enkripsi tetapi xor dilakukan pada naskah acak, dengan kunci yang sama (kunci dekripsi sama dengan kunci enkripsi). Setiap bit dalam kunci jika dioperasikan terhadap bit dalam naskah asli (seperti dalam proses enkripsi) kemudian dioperasikan lagi terhadap hasil operasi pertama (seperti dalam proses dekripsi) akan mendapatkan kembali bit naskah asli. Ini adalah konsekuensi sifat aljabar operasi xor.

bit naskah	bit kunci	bit hasil operasi
0	0	0
1	0	1
0	1	1
1	1	0

Tabel 2.3: Tabel operasi xor

Tabel 2.3 memperlihatkan operasi xor yang digunakan oleh *one-time pad*. Nilai 0 untuk bit kunci mempertahankan nilai bit naskah yang dioperasikan, jadi dua kali mempertahankan akan tetap mempertahankan nilai bit naskah asli. Nilai 1 untuk bit kunci menghasilkan negasi bit naskah yang dioperasikan, jadi dua kali negasi akan mendapatkan nilai bit semula yaitu nilai bit naskah asli. Alhasil, nilai apapun untuk bit kunci akan mendapatkan kembali nilai bit naskah asli jika dioperasikan dua kali. Operasi *exclusive or* sangat berperan dalam kriptografi: semua algoritma enkripsi simetris modern menggunakan operasi *exclusive or*. Simbol $\oplus$ kerap digunakan sebagai notasi untuk *exclusive or*.

Ketangguhan enkripsi *one-time pad* tergantung pada keacakan kunci yang digunakan. *Key management* menjadi sangat penting dan merupakan kelemahan *one-time pad* yang membuatnya tidak layak untuk penggunaan skala besar[2]. Besar kunci harus menyamai besar naskah asli, dan kunci tidak boleh diguna-ulang. Selain masalah *key generation*, masalah *key distribution* menjadi kendala penggunaan skala besar enkripsi *one-time pad*.

Secara historis, enkripsi *one-time pad* digunakan oleh misi diplomatik berbagai negara di masa lalu untuk komunikasi rahasia. Semacam buku kode yang dibuat secara acak dan tidak boleh diguna-ulang harus dibawa oleh kurir yang dipercaya untuk didistribusikan ke perwakilan diplomatik negara yang bersangkutan. Setiap pengiriman naskah rahasia, kunci sebesar naskah rahasia diambil dari buku kode untuk mengenkripsi naskah. Kunci yang sudah digunakan untuk enkripsi tidak boleh digunakan lagi untuk enkripsi selanjutnya.

One-time pad dapat digunakan untuk komunikasi sangat rahasia dengan volume yang tidak terlalu besar, namun untuk penggunaan skala besar dalam suatu sistem teknologi informasi, *one-time pad* tidak praktis. Walaupun tidak digunakan secara langsung, konsep *one-time pad* "ditiru" dalam teknik enkripsi *stream cipher* (lihat bab 6).

[2]Situasi ini dapat berubah jika *quantum key distribution* menjadi sesuatu yang praktis.

2.3 Cryptanalysis

Cryptanalysis adalah teknik untuk mencoba memecahkan enkripsi, biasanya dengan mencari kunci enkripsi. Ada tiga kategori teknik pencarian kunci yang biasanya digunakan untuk kriptografi klasik yaitu

- *known plaintext attack*,
- analisa statistik, dan
- *brute force search*.

Kombinasi dari teknik-teknik diatas juga kerap digunakan. Biasanya minimal pemecah mempunyai akses ke naskah acak, dan kadang juga mengetahui naskah aslinya. Kita akan bahas ketiga teknik tersebut.

Algoritma enkripsi klasik yang baik adalah algoritma yang tahan terhadap *known plaintext attack* dan analisa statistik sehingga pencarian kunci harus dilakukan dengan *brute force search*. Tentunya kunci enkripsi harus cukup besar agar *brute force search* tidak efektif.

Berbeda dengan kriptografi klasik, kriptografi *public key* mengandalkan keamanannya pada sukarnya komputasi untuk mendapatkan kunci rahasia, yaitu

- penguraian bilangan bulat yang besar, atau
- komputasi logaritma diskrit untuk *finite field* yang besar.

Jadi pemecahan kunci untuk kriptografi *public key* difokuskan pada teknik-teknik untuk mempercepat kedua komputasi tersebut. Kita akan bahas penguraian bilangan bulat di bab 14 dan logaritma diskrit di bab 15.

2.3.1 Known Plaintext Attack

.

Known plaintext attack adalah teknik pencarian kunci enkripsi berdasarkan pengetahuan mengenai pasangan naskah asli - naskah acak. Kita akan gunakan *Caesar cipher* sebagai contoh dari enkripsi yang rentan terhadap *known plaintext attack*.

Julius Caesar menukar setiap huruf dalam naskah asli dengan huruf lain dalam naskah acak. Besar atau kecil huruf dipertahankan dalam naskah acak (huruf besar ditukar dengan huruf besar, huruf kecil ditukar dengan huruf kecil). Spasi, titik, koma dan tanda lainnya tidak ditukar.

Caesar cipher adalah jenis enkripsi yang disebut *simple substitution cipher* dimana setiap huruf dalam naskah asli ditukar dengan huruf lain dalam naskah

acak. Julius Caesar menukar huruf dengan cara *shift transformation*. Rumus umum untuk *shift transformation* adalah:

$$C = \begin{cases} P + b & \text{jika } P + b < n \\ P + b - n & \text{jika } P + b \geq n \end{cases} \tag{2.1}$$

dimana:
C adalah kode bilangan karakter acak,
P adalah kode bilangan karakter asli,
b adalah besarnya *shift*,
n adalah besarnya perbendaharaan karakter (dengan kode 0 sampai $n - 1$).

Jadi rumus untuk enkripsi sesuai dengan relasi ekuivalen:

$$C \equiv P + b \pmod{n}. \tag{2.2}$$

Rumus untuk dekripsi juga sesuai dengan relasi ekuivalen 2.2:

$$P = \begin{cases} C - b & \text{jika } C \geq b \\ C - b + n & \text{jika } C < b. \end{cases} \tag{2.3}$$

Julius Caesar sendiri menggunakan huruf "A" sampai "Z" (dengan kode 0 sampai 25) sebagai perbendaharaan karakter untuk enkripsi (karakter selain huruf tidak dienkripsi), dan menggunakan parameter $b = 3$ menghasilkan rumus enkripsi

$$C = \begin{cases} P + 3 & \text{jika } P < 23 \\ P - 23 & \text{jika } P \geq 23. \end{cases} \tag{2.4}$$

Jadi untuk enkripsi, 0 ("A") ditukar dengan 3 ("D"), 1 ("B") dengan 4 ("E"), ..., 24 ("Y") dengan 1 ("B"), dan 25 ("Z") dengan 2 ("C").

Rumus dekripsi menjadi

$$P = \begin{cases} C - 3 & \text{jika } C \geq 3 \\ C + 23 & \text{jika } C < 3. \end{cases} \tag{2.5}$$

Naskah Asli	`Jangan rahasiakan pesan ini!`
Naskah Acak	`Mdqjdq udkdvldndq shvdq lql!`

Tabel 2.4: Enkripsi dengan *Caesar cipher*

Enkripsi yang menggunakan *shift transformation* seperti *Caesar cipher* sangat rentan terhadap *known plaintext attack*. Jika pasangan naskah asli - naskah acak diketahui, parameter b dapat ditemukan dengan mudah. Sebagai

contoh, jika pasangan dalam tabel 2.4 diketahui, kita dapat menggunakan pasangan huruf acak - huruf asli berdasarkan posisi, misalnya "M" dengan "J". Ini akan segera mendapatkan $b = (12 - 9) \bmod 26 = 3$.

Known plaintext attack terhadap enkripsi *Caesar cipher* adalah contoh *attack* yang bersifat deterministik dimana jika pasangan (atau beberapa pasangan) naskah asli - naskah acak diketahui maka kunci dapat ditemukan dengan pasti. Jika tidak hati-hati, enkripsi *one-time pad* juga sangat rentan terhadap *known plaintext attack* yang bersifat deterministik. Operasi *exclusive or* terhadap naskah asli dan naskah acak langsung mendapatkan kunci. Oleh karena itu kunci untuk *one-time pad* tidak boleh diguna-ulang (itulah maksud nama *one-time pad*: hanya digunakan satu kali). Efek serupa berlaku untuk enkripsi yang "meniru" *one-time pad* seperti *stream cipher* (dimana operasi *exclusive or* terhadap naskah acak dan naskah asli langsung mendapatkan *keystream*), jadi penggunaan *stream cipher* harus dengan sangat hati-hati.

Tidak semua *known plaintext attack* bersifat deterministik. *Linear cryptanalysis* (lihat bagian 8.2) menggunakan *known plaintext attack* tetapi secara probabilistik.

Tingkat kesukaran *known-plaintext attack* tergantung pada rumus yang digunakan untuk enkripsi. Semakin rumit rumus yang digunakan, semakin sukar untuk melakukan *known-plaintext attack*. Rumus yang bersifat linear masih tergolong sederhana dan dianggap rentan terhadap *known-plaintext attack*. Semakin non-linear dan semakin banyak parameter yang digunakan untuk rumus, semakin sulit untuk mengkalkulasi kunci berdasarkan rumus. Ini akan semakin jelas dengan pembahasan berbagai macam enkripsi di bab-bab selanjutnya.

2.3.2 Analisa Statistik

Kecuali *one-time pad*, semua algoritma enkripsi sebelum *Data Encryption Standard* (DES) rentan terhadap analisa statistik. Sebagai contoh, mari kita lihat bagaimana enkripsi dengan cara *shift transformation* seperti *Caesar cipher* rentan terhadap analisa statistik yang sederhana yaitu analisa frekuensi.

Enkripsi dengan cara *shift transformation* sangat rentan terhadap analisa frekuensi sebagai berikut: dengan rumus enkripsi

$$C \equiv P + b \pmod{n},$$

jika n diketahui dan sepasang C dan P dapat diterka dengan akurat, maka parameter b (kunci) dapat dicari. Setiap "pencarian" b dapat dicoba cukup dengan sepasang nilai untuk C dan P. Pasangan nilai yang patut dicoba adalah pasangan yang sesuai dengan statistik frekuensi penggunaan. Sebagai contoh, menggunakan naskah acak dalam tabel 2.4, huruf "D" dan "Q" adalah yang terbanyak digunakan dalam naskah acak. Karena dalam bahasa Indonesia, huruf "A" adalah huruf dengan statistik penggunaan terbesar, jika naskah

asli dalam bahasa Indonesia, maka besar kemungkinan huruf "D" atau "Q" merupakan huruf acak untuk "A". Jadi besar kemungkinan, jika kita menggunakan kode untuk "D" atau "Q" sebagai nilai C dan kode untuk "A" sebagai nilai P, rumus enkripsi akan menghasilkan nilai b yang benar. Jadi kita coba dua kemungkinan: pasangan "D-A" (yang menghasilkan $b = 3$) dan pasangan "Q-A" (yang menghasilkan $b = 16$). Hasil yang dicari adalah nilai b yang jika digunakan untuk mendekripsi naskah acak akan menghasilkan naskah asli yang "masuk akal."

Pasangan	Kode Acak	Kode Asli	Nilai b	Hasil Dekripsi
D-A	3	0	$b = 3$	`Jangan rahasiakan pesan ini!`
Q-A	16	0	$b = 16$	`Wnatna enunfvnxna crfna vav!`

Tabel 2.5: Hasil analisa frekuensi

Berdasarkan hasil analisa frekuensi, yang "masuk akal" hanya $b = 3$ dengan hasil dekripsi "Jangan rahasiakan pesan ini!", jadi kita dapat cukup yakin bahwa parameter $b = 3$.

Analisa frekuensi diatas didasarkan pada pengetahuan bahwa naskah asli adalah dalam bahasa Indonesia dimana huruf "A" mempunyai statistik frekuensi penggunaan terbesar[3]. Tentunya naskah asli tidak akan selalu mempunyai statistik yang mirip dengan data empiris. Tetapi secara umum, semakin panjang naskah yang digunakan untuk analisa, semakin besar kemungkinan statistik penggunaan akan mirip dengan data empiris, yang berarti semakin besar kemungkinan analisa frekuensi akan sukses. Dalam contoh diatas, statistik penggunaan huruf "A" cukup mirip dengan data empiris, jadi analisa frekuensi berhasil.

Untuk *shift transformation*, karena rumus transformasi sangat sederhana hanya dengan satu parameter, setiap percobaan cukup dengan menggunakan satu persamaan. Strategi pencarian yang baik adalah dengan mencoba pasangan yang mempunyai frekuensi penggunaan yang besar (huruf acak yang frekuensinya besar dalam naskah acak dipasangkan dengan huruf asli yang frekuensinya juga besar menurut data empiris). Semakin besar frekuensi terbesar dalam data empiris, secara umum berarti semakin besar *redundancy* dari segi teori informasi, yang akan mempermudah analisa frekuensi.

Jika rumus transformasi lebih rumit dengan lebih dari satu parameter, maka setiap percobaan harus dilakukan dengan lebih dari satu persamaan, dengan banyaknya persamaan yang dibutuhkan sedikitnya sama dengan banyaknya parameter yang harus dicari.

[3]Untuk bahasa Inggris, frekuensi terbesar adalah untuk huruf "E" dan statistik penggunaan untuk semua huruf cukup diketahui berdasarkan pengamatan empiris.

Untuk sukses dalam analisa frekuensi, dibutuhkan pengetahuan empiris mengenai statistik penggunaan huruf, naskah acak yang dapat dianalisa harus cukup besar, rumus atau seminimnya jenis enkripsi harus diketahui (jika rumus tidak diketahui tetapi jenis enkripsi diketahui berupa *simple substitution*, setiap huruf acak harus dipasangkan dengan huruf asli). Untuk analisa frekuensi yang rumit, penggunaan komputer sangat membantu.

Pada umumnya, semakin besar data yang digunakan sebagai dasar, baik untuk probabilitas *a priori* seperti frekuensi penggunaan huruf, maupun yang bersifat *a posteriori* yaitu naskah acak, semakin pasti (tidak tentatif) hasil percobaan kalkulasi. Sebagai contoh, untuk *shift transformation* dengan perbendaharaan 26 huruf dan naskah dalam bahasa Inggris, analisa frekuensi dengan naskah acak sebesar 50 karakter atau lebih akan mendapatkan hasil dengan kepastian mendekati 100 persen, berdasarkan pengetahuan *a priori* mengenai penggunaan huruf dalam bahasa Inggris.

Secara umum, enkripsi yang rentan terhadap *known plaintext attack* yang deterministik juga rentan terhadap analisa frekuensi, jika data empiris mengenai statistik naskah asli diketahui. Ini karena analisa frekuensi dapat dipandang sebagai suatu *known plaintext attack* yang probabilistik, dimana pasangan naskah asli - naskah acak diterka atau diperkirakan berdasarkan data empiris.

Analisa frekuensi lebih mudah dilakukan pada *substitution cipher* dibandingkan dengan *block cipher*. Enigma berhasil dipecahkan oleh pihak sekutu menggunakan analisa frekuensi karena enkripsi yang digunakan Enigma adalah *substitution cipher*, meskipun jenisnya adalah *polyalphabetic* (pertukaran huruf berubah terus). Analisa frekuensi terhadap *polyalphabetic substitution cipher* memang jauh lebih sukar dibandingkan dengan analisa frekuensi terhadap *simple substitution cipher*, tetapi jauh lebih mudah dibandingkan analisa frekuensi terhadap *block cipher*. Fakta ini serta beberapa kelemahan Enigma lainnya, seperti tidak pernah menukar huruf dengan huruf yang sama, berhasil dieksploitasi oleh pihak sekutu dibawah pimpinan Alan Turing.

Analisa frekuensi merupakan contoh dari analisa statistik yang tergolong sederhana. Kita akan bahas analisa statistik yang lebih canggih yaitu *linear cryptanalysis* dan *differential cryptanalysis* di bab 8 setelah kita bahas *block cipher*.

2.3.3 Brute Force Search

Satu dari kriteria sistem enkripsi yang baik adalah bahwa dekripsi tanpa kunci hanya dapat dipecahkan dengan cara *brute force search* dimana semua kemungkinan kunci harus dicoba. Tentunya jumlah kemungkinan kunci harus cukup besar sehingga diperlukan waktu yang sangat lama untuk mencoba semua kunci. Jika jumlah kunci yang harus dicoba kurang besar, maka sistem enkripsi rentan terhadap analisa *brute force search*.

Besarnya kunci enkripsi (jumlah bit dalam kunci enkripsi) menentukan jumlah kemungkinan kunci yang harus dicoba dalam *brute force search*. Untuk kunci sebesar n *bits*, jumlah kemungkinan kunci adalah 2^n dan rerata, kunci akan ditemukan setelah kita mencoba 2^{n-1} kemungkinan (setengah dari semua kemungkinan). Jadi enkripsi rentan terhadap *brute force search* jika 2^n kemungkinan kunci dapat dicoba dalam waktu yang tidak terlalu lama. Tentunya selain tergantung pada jumlah kemungkinan kunci yang harus dicoba, waktu yang diperlukan juga tergantung pada kemampuan hardware yang digunakan. Juga batas "waktu yang tidak terlalu lama" tergantung pada aplikasi, apakah kurang dari 1 bulan, kurang dari 5 tahun, kurang dari 1000 tahun, atau ada batas waktu lain.

Caesar cipher, selain dapat dipecahkan dengan analisa frekuensi atau *known plaintext attack*, dapat juga dipecahkan dengan *brute force search* karena semua kemungkinan kunci ($b = 1$ sampai dengan $b = 25$) dapat dicoba dalam waktu yang tidak terlalu lama. Kita gunakan contoh naskah acak yang digunakan dalam analisa frekuensi (lihat 2.3.2), yaitu "`Mdqjdq udkdvldndq shvdq lql!`".

Tabel 2.6 memperlihatkan hasil *brute force search* terhadap naskah acak *Caesar cipher*. Dari semua kemungkinan kunci, hanya $b = 3$ yang "masuk akal." Karena jumlah kemungkinan kunci cukup kecil, *brute force search* dapat dilakukan tanpa menggunakan komputer. Jika jumlah kunci yang harus dicoba sangat besar, komputer dapat digunakan untuk membantu analisa.

Besar kunci enkripsi ikut menentukan sukses dari *brute force search*. Dengan kemajuan dibidang hardware untuk melakukan *brute force search* (hardware khusus dapat dibuat untuk melakukan *brute force search* terhadap kunci *block cipher*), enkripsi *block cipher* dengan besar kunci 56 bit (jadi ada 2^{56} kemungkinan) kini dapat dipecahkan dalam waktu yang tidak terlalu lama (kira-kira puluhan menit) dengan hardware seharga 1 juta USD. Dengan hardware yang lebih banyak, waktu yang diperlukan menjadi semakin singkat dengan perbandingan waktu *inverse proportional* terhadap harga. Untuk keamanan, sebaiknya enkripsi *block cipher* menggunakan kunci minimum 128 bit. *Data Encryption Standard* (DES) hanya menggunakan 56 bit untuk kunci, jadi sebaiknya diganti dengan 3DES atau *block cipher* lain seperti CAST, AES dan Blowfish.

2.4 Manajemen Kunci

Aspek manajemen kunci sangat penting dalam aplikasi kriptografi. Manajemen kunci yang tidak baik dapat berakibat fatal. Konon[4], di masa perang dingin, pihak Uni Soviet "kecurian" rahasia penting oleh pihak Amerika Serikat

[4]Tidak diverifikasi oleh penulis.

Nilai b	Hasil Dekripsi
$b = 1$	`Lcpicp tcjcukcmcp rgucp kpk!`
$b = 2$	`Kbohbo sbibtjblbo qftbo joj!`
$b = 3$	`Jangan rahasiakan pesan ini!`
$b = 4$	`Izmfzm qzgzrhzjzm odrzm hmh!`
$b = 5$	`Hyleyl pyfyqgyiyl ncqyl glg!`
$b = 6$	`Gxkdxk oxexpfxhxk mbpxk fkf!`
$b = 7$	`Fwjcwj nwdwoewgwj laowj eje!`
$b = 8$	`Evibvi mvcvndvfvi kznvi did!`
$b = 9$	`Duhauh lubumcueuh jymuh chc!`
$b = 10$	`Ctgztg ktatlbtdtg ixltg bgb!`
$b = 11$	`Bsfysf jszskascsf hwksf afa!`
$b = 12$	`Arexre iryrjzrbre gvjre zez!`
$b = 13$	`Zqdwqd hqxqiyqaqd fuiqd ydy!`
$b = 14$	`Ypcvpc gpwphxpzpc ethpc xcx!`
$b = 15$	`Xobuob fovogwoyob dsgob wbw!`
$b = 16$	`Wnatna enunfvnxna crfna vav!`
$b = 17$	`Vmzsmz dmtmeumwmz bqemz uzu!`
$b = 18$	`Ulyrly clsldtlvly apdly tyt!`
$b = 19$	`Tkxqkx bkrkcskukx zockx sxs!`
$b = 20$	`Sjwpjw ajqjbrjtjw ynbjw rwr!`
$b = 21$	`Rivoiv zipiaqisiv xmaiv qvq!`
$b = 22$	`Qhunhu yhohzphrhu wlzhu pup!`
$b = 23$	`Pgtmgt xgngyogqgt vkygt oto!`
$b = 24$	`Ofslfs wfmfxnfpfs ujxfs nsn!`
$b = 25$	`Nerker velewmeoer tiwer mrm!`

Tabel 2.6: Mencoba semua kemungkinan kunci *Caesar cipher*

akibat masalah dalam manajemen kunci. Ternyata pihak Uni Soviet, karena masalah kurangnya dana, mengguna-ulang kode untuk enkripsi *one-time pad*. Pihak Amerika Serikat berhasil menggunakan kesalahan ini untuk memecahkan sebagian komunikasi rahasia pihak Uni Soviet.

Proses pembuatan kunci sangat penting dan sebaiknya proses ini benar acak. Sumber acak (entropi) dapat diambil dari proses fisika acak seperti proses radio-aktif. Sumber acak dapat juga diambil dari berbagai kejadian (*events*) yang muncul secara acak. *Operating system* seperti unix menggunakan kombinasi *system events* termasuk *interrupts* sebagai sumber entropi[5], yang

[5]Dalam suatu *operating system* biasanya pembuatan entropi menggunakan kombinasi dari banyak sumber agar entropi yang cukup besar didapatkan dalam jangka waktu yang cukup

kemudian dikombinasikan dengan algoritma *pseudo-random number generator* menjadi *random number generator*. Aplikasi kriptografi dapat menggunakan *random number generator* yang disediakan *operating system* untuk pembuatan kunci, akan tetapi sebaiknya ini dilakukan hanya jika *random number generator* yang disediakan cukup acak.

Distribusi kunci secara aman juga penting untuk keperluan pengamanan komunikasi. Sebagai contoh, untuk komunikasi yang diamankan dengan enkripsi simetris, tentunya kedua mitra dalam komunikasi harus menggunakan kunci yang sama. Kunci ini dapat dibuat oleh satu pihak dan dikirim secara aman ke mitra komunikasi. Pengiriman kunci dapat dilakukan *out-of-band* yaitu menggunakan jalur khusus diluar jalur normal komunikasi, atau dilakukan *in-band* melalui jalur normal menggunakan sarana *public key cryptography*. Alternatif dari pengiriman kunci adalah *key agreement*, dimana kedua mitra berpartisipasi membuat kunci tanpa dapat diketahui oleh pihak ketiga. *Key agreement* juga menggunakan sarana *public key cryptography*.

Penyimpanan kunci jelas sangat penting untuk pengamanan sistem enkripsi secara menyeluruh. Kunci yang disimpan secara sembrono akan mudah untuk "dicuri" oleh pihak yang tidak diinginkan. Solusi untuk penyimpanan kunci beraneka ragam, mulai dari penggunaan hardware khusus dimana semua proses kriptografi dilakukan didalam hardware khusus dan kunci enkripsi disimpan dan tidak dapat keluar dari hardware, sampai dengan penyimpanan dalam *file* yang dienkripsi menggunakan password atau *passphrase*. Karena praktis, metode terahir sangat populer, yang berarti pengamanan password menjadi penting.

Pengamanan password juga mempunyai beberapa masalah, dari masalah manusia seperti menulis password di secarik kertas yang ditempelkan ke meja kerja, sampai dengan masalah sistem seperti program yang menyimpan password dalam bentuk teks.

Pada dasarnya masalah akses terhadap sesuatu yang penting seperti kunci enkripsi menjadi masalah *authentication* dan tren saat ini mengarah pada *multiple factor authentication*. Kebenaran identititas seseorang atau sesuatu dinilai dari gabungan berbagai atribut yang cukup unik seperti sidik jari, pengetahuan password, dan kepemilikan sesuatu yang unik lainnya.

2.5 Operasi dasar

Operasi terpenting terhadap unit data dalam kriptografi adalah *exclusive or* (xor), seperti yang digunakan dalam enkripsi *one-time pad*. Operasi xor sangat mudah implementasinya dalam hardware, dan prosesor komputer biasanya memiliki instruksi untuk melakukan *bitwise* xor.

singkat.

Jika *one-time pad* dapat digunakan dalam skala besar, maka enkripsi hanya memerlukan operasi xor. Akan tetapi, *one-time pad* tidak praktis untuk penggunaan skala besar, oleh sebab itu diperlukan operasi lainnya yaitu substitusi dan permutasi.

Substitusi adalah proses penukaran unit data secara utuh, seperti yang dilakukan dalam *Caesar cipher* dimana huruf ditukar dengan huruf. Operasi ini membuat efek *confusion* (pembingungan) terhadap analisa statistik. Spesifikasi dan implementasi operasi ini dapat dilakukan menggunakan tabel jika tabel tidak terlalu besar, dengan nilai yang hendak ditukar digunakan sebagai indeks tabel, dan nilai dalam tabel digunakan sebagai nilai penukar. Contoh dari operasi substitusi menggunakan tabel adalah operasi substitusi S1 yang digunakan algoritma DES, seperti terlihat pada tabel 2.7. Tabel mempunyai 4 baris (dengan indeks 0, 1, 2, 3) dan 16 kolom (dengan indeks 0 sampai dengan 15). Input 6 bit digunakan sebagai indeks baris dan kolom, dengan bit 1 dan bit 6 menentukan indeks baris dan bit 2 sampai dengan bit 5 menentukan indeks kolom. Sebagai contoh, jika input 6 bit adalah 011011, maka indeks baris adalah 1 (karena bit 1, 6 mempunyai nilai 01), dan indeks kolom adalah 13 (karena bit 2, 3, 4, 5 mempunyai nilai 1101). Dengan input 011011, S1 akan menghasilkan 4 bit 0101 karena baris 1 kolom 13 dalam tabel S1 mempunyai nilai 5, yang dalam notasi biner 4 bit adalah 0101.

S1

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

Tabel 2.7: Tabel untuk S1

Jika tabel terlalu besar, biasanya operasi substitusi mempunyai rumus untuk mengkalkulasi nilai tukar, seperti rumus enkripsi *Caesar cipher* dan rumus untuk S-box AES. *Caesar cipher* dan S-box untuk AES, meskipun mempunyai rumus yang elegan, dapat juga diimplementasikan menggunakan tabel karena tabel tidak terlalu besar. Akan tetapi, S-boxes untuk DES tidak memiliki rumus yang elegan, jadi biasanya diimplementasikan menggunakan tabel.

Permutasi adalah proses penukaran posisi dalam unit data. Operasi ini membuat efek *diffusion* (pembauran) yang mempersulit analisa statistik. Operasi ini dapat diimplementasikan dalam hardware secara efisien. Spesifikasi operasi permutasi dan implementasi dengan software biasanya dilakukan menggunakan tabel. Posisi awal komponen data digunakan sebagai indeks tabel, dan nilai dalam tabel digunakan sebagai posisi ahir komponen data. Operasi permutasi bersifat *bijective map* dimana tidak ada komponen data yang hilang

dan tidak ada komponen data yang digandakan. Dalam kriptografi, komponen data dalam operasi permutasi biasanya berupa bit. Contoh operasi permutasi adalah *Initial Permutation* yang digunakan algoritma DES, seperti terlihat dalam tabel 2.8. Posisi ahir bit digunakan sebagai indeks tabel, jadi nilai output bit n sama dengan nilai input bit $T(n)$ dimana $T(n)$ adalah nilai komponen n dalam tabel. Sebagai contoh, nilai output bit 1 sama dengan nilai input bit 58, karena komponen pertama dalam tabel mempunyai nilai 58. Permutasi bit dapat diimplementasikan dalam hardware secara efisien karena hanya dibutuhkan koneksi antara posisi awal bit dengan posisi ahir bit. Jadi untuk *Initial Permutation* hanya diperlukan 64 koneksi, dengan bit 58 input menjadi bit 1 output, bit 50 input menjadi bit 2 output, dan seterusnya.

IP

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

Tabel 2.8: Tabel *Initial Permutation*

2.6 Ringkasan

Dalam bab ini, pembaca telah diperkenalkan dengan konsep enkripsi dan dekripsi meskipun hanya secara garis besar. Pembaca juga diperkenalkan dengan konsep acak, suatu konsep yang sangat penting dalam kriptografi. Teknik enkripsi *one-time pad* merupakan teknik enkripsi "sempurna" dari segi teoritis, meskipun dalam prakteknya banyak kendala penggunaannya. Kita melihat bagaimana analisa statistik dapat dipergunakan untuk memecahkan enkripsi yang kurang kuat. Pemecahan enkripsi juga dapat dipermudah jika kita mengetahui naskah asli atau bagian dari naskah asli. Selain analisa statistik, pemecahan enkripsi juga dapat dilakukan dengan *brute force search* jika ruang pencarian relatif tidak terlalu besar. Tidak kalah pentingnya dalam kriptografi adalah masalah manajemen kunci, karena jika tidak hati-hati kunci dapat jatuh ke tangan pihak yang tidak diinginkan. Yang terahir, pembaca diperkenalkan dengan operasi dasar yang banyak digunakan dalam kriptografi. Konsep-konsep yang diperkenalkan dalam bab ini akan digunakan pada bab-bab selanjutnya.