PENERAPAN ALGORITMA HUFFMAN DALAM DUNIA KRIPTOGRAFI

Yogie Adrisatria – NIM : 13503035

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13035@students.if.itb.ac.id

Abstraksi

Tujuan dari proses kompresi data adalah untuk menjadikan ukuran data lebih kecil dari aslinya. Untuk mencapai tujuan tersebut, diperlukan transformasi atau perubahan antara data asli dengan data hasil kompresi. Hasil transformasi tersebut biasanya berbentuk sesuatu yang cukup berbeda dengan data asalnya dan untuk mendapatkan datanya kembali diperlukan beberapa data yang berkaitan dengan data aslinya.

Dilihat dari hal tersebut, proses transformasi dalam bidang kompresi data dapat dianalogikan dengan proses enkripsi dalam bidang kriptografi. Algoritma Huffman sebagai salah satu algoritma kompresi data yang cukup tua memiliki prinsip-prinsip yang dapat diterapkan di bidang enkripsi data walaupun harus dilakukan beberapa adaptasi, terutama menyangkut implementasinya agar lebih praktis. Makalah ini akan membahas penerapan algoritma Huffman dalam dunia kriptogafi untuk akhirnya diambil sebuah kesimpulan apakah algoritma Huffman ini cukup layak dibahas sebagai algoritma kriptografi dilihat dari segi keamanan dan kepraktisannya.

Kata Kunci: *Data Compression*, Huffman, Kriptografi.

1. Pendahuluan

Kriptografi adalah bidang ilmu yang bertujuan untuk menjamin keamanan suatu data agar tidak diketahui dan tidak dimengerti oleh orang lain yang tidak berhak. Untuk mencapai tujuan tersebut hal yang dilakukan adalah mengenkripsi pesan sedemikian rupa sehingga *plaintext* yang tadinya bisa terbaca menjadi sebuah teks lain yang tidak dapat terbaca atau disebut juga *ciphertext*. Agar orang yang berhak membaca teks tersebut dapat mengetahui isi sebenarnya dari *ciphertext*, diperlukan sebuah proses untuk mengembalikan *ciphertext* kembali menjadi sebuah *plaintext* yang disebut sebagai proses dekripsi. Untuk melakukan proses enkripsi dan dekripsi tersebut dibutuhkan sebuah kunci yang hanya diketahui oleh pengirim dan penerima. Bila kunci yang untuk melakukan enkripi dan dekripsi adalah kunci yang sama, algoritma kriptografi disebut sebagai sebuah algoritma kriptografi simetris.

Sementara kompresi data (*data compression*) merupakan salah satu bidang Teori Informasi (*Information Theory*) yang bertujuan untuk melakukan pemampatan data sehingga perepresentasian data dapat dilakukan dengan ukuran yang lebih kecil. Hal ini sangat berguna ketika data tersebut perlu disimpan dalam tempat yang terbatas ataupun data akan dikirim melalui sebuah saluran komunikasi yang memiliki *bandwidth* terbatas. Aplikasi data kompresi sekarang ini telah dikenal dengan sangat luas karena telah memberikan berbagai kemudahan. Salah satu hal yang dilakukan dalam algoritma untuk melakukan kompresi data adalah melakukan transformasi pesan menjadi sesuatu bentuk yang berbeda yang tentunya akan lebih kecil dari ukuran asilnya. Salah satu ukuran bagus tidaknya data hasil kompresi adalah tingkat keacakan dari data tersebut. Makin tinggi tinggi tingkat *randomness*-nya, maka makin tinggi pula tingkat kompresinya.

Proses tranformasi dalam proses kompresi data dapat dianalogikan dengan proses enkripsi dalam dunia kriptografi. Begitu pula dengan proses dekompresi data, dapat dianalogikan dengan proses dekripsi data dalam kriptografi. Sementara kunci untuk kriptografi didapatkan dari informasi yang dibutuhkan algoritma kompresi dalam melakukan dekompresi.

Beberapa algoritma kompresi data yang telah dicoba dimanfaatkan dalam bidang kriptografi antara lain:

- Algoritma Lampel-Ziv

- Algoritma Huffman
- Algoritma Shannon-Fano-Elias
- Algoritma *Arithmetic Coding*

Salah satu aplikasi dari penggunaan algoritma kompresi untuk kriptografi adalah untuk pengiriman data multimedia karena sifat dari algoritma kompresi yang akan sekaligus melakukan pemampatan data [6]. Selain itu, hal yang melatarbelakangi penggunaan algoritma kompresi untuk pengenkripsian adalah untuk penyimpanan *back-up* suatu basis data yang sangat besar dan juga membutuhkan pengamanan atas data yang ada [4].

2. Algoritma Kompresi Data

Bidang kompresi data merupakan salah satu bidang yang fundamental dalam Teori Informasi (*Information Theory*). Teori Informasi sendiri merupakan cabang dari ilmu matematika yang lahir pada akhir era 1940-an melalui kerja dari Claude Shannon di Laboratorium Bell. Bidang ini merupakan bidang yang berkecimpung dalam hal mengenai pengeloaan infomasi, termasuk di dalamnya adalah bagaimana cara menyimpan dan mengkomunikasikan pesan.

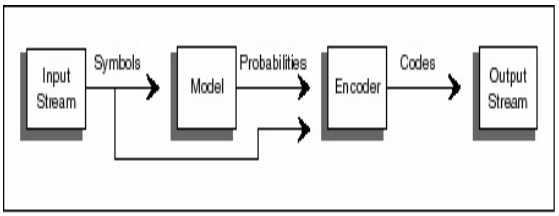
Kompresi data menjadi salah satu cabang Teori Informasi karena kompresi data berkecimpung dengan masalah redundansi dalam suatu informasi. Informasi yang redundan memakan bit ekstra untuk mengkodekannya dan jika bit ekstra tersebut dapat dihilangkan berarti telah terjadi pengurangan ukuran informasi.

Dunia data kompresi sendiri dapat dibagi menjadi dua, yaitu *lossy data compression* dan *lossless data compression*. *Lossy data compression* adalah teknik kompresi yang membolehkan terjadinya kehilangan beberapa bit data dengan imbal balik berupa tingkat kompresi yang cukup tinggi. Contoh dari teknik kompresi ini banyak diimplementasikan dalam dunia multimedia, seperti pemampatan file gambar (jpg, gif, png, dan lain-lain) dan file video (mpg, avi, rmvb). Algoritma *lossy* tersebut dapat diterapkan dalam bidang multimedia karena file-file multimedia biasanya masih dapat dinikmati walaupun ada beberapa bit yang hilang, walaupun tentunya ada sedikit penurunan kualitas.

Sementara itu, algoritma *lossless data compression* adalah teknik kompresi data yang tidak membolehkan terjadinya kehilangan data baik dalam proses kompresi maupun proses dekompresi. Aplikasi dari *lossless data compression* sekarang ini sangatlah luas, seperti dalam aplikasi-aplikasi *archiving* seperti WinZip, WinRar, dan WinAce.

Contoh dari algoritma *lossless data compression* cukup banyak. Contoh-contoh klasik dari algoritma jenis ini antara lain algoritma Shannon-Fano dan algoritma Huffman yang akan dibahas dalam makalah ini.

Secara umum, proses kompresi data terdiri dari proses mengambil aliran (*stream*) dari simbol-simbol dan mentransformasikannya menjadi kode-kode. Jika tingkat kompresinya cukup efektif, *stream* keluarannya akan memiliki ukuran yang lebih kecil dari *stream* masukannya. Keputusan untuk meng-*output*-kan suatu kode didasarkan pada sebuah model. Model sendiri adalah sebuah koleksi data dan aturan yang memproses simbol-simbol masukan dan menentukan kode yang harus dioutpukan. Program kompresi menggunakan model untuk mendefinisikan secara akurat setiap simbol dan kode yang akan dihasilkan yang didasarkan beberapa probabilitas. Garis besar proses kompresi data dapat dilihat di gambar di bawah ini.



Gambar 1 Proses Kompresi Data

3. Algoritma Huffman

Algoritma Huffman adalah salah satu algoritma kompresi yang sudah cukup tua tetapi tetap dinilai sebagai salah satu algoritma kompresi data yang handal. Algoritma Huffman ini pertama kali diterbitkan pada tahunn 1952 oleh D.A. Huffman dalam *paper*-nya yang berjudul "*A Method for the Construction of Minimum Redundancy Codes*". Secara umum, algoritma ini dapat memberikan penghematan sebesar 20%-30%. Algoritma Huffman dapat diklasifikasikan sebagai algoritma *lossless data compression* karena tidak boleh ada bit yang hilang selama proses kompresi maupun dekompresi.

Prinsip kerja dari Algoritma Huffman adalah mengodekan setiap karakter ke dalam representasi bit. Representasi bit untuk setiap karakter akan berbeda dari segi panjangnya. Hal ini disebut sebagai *variable-length code*. Panjang pendeknya representasi bit bagi sebuah karakter ditentukan oleh frekuensi kemunculan karakter tersebut dalam *plaintext*. Makin sering suatu karakter muncul, makin pendek panjang bit yang merepresentasikannya. Sebaliknya, makin jarang suatu karakter muncul, akan makin panjang reprsententasi bitnya. Prinsip ini sangat bermanfaat dalam melakukan kompresi karena pada umumnya pada sebuah berkas terdapat berbagai perulangan dan pola. Perulangan dan pola inilah yang dimanfaatkan oleh algoritma Huffman untuk melakukan kompresi data.

Sebagai contoh, misalnya ada sebuah berkas yang berisi 100.000 karakter. Andaikan data di dalam berkas disusun oleh karakter *a,b,c,d,e,* dan *f* dengan frekuensi kemunculan huruf-hurufnya sebagai berikut:

Tabel 1 Frekuensi huruf dalam sebuah berkas

Karakter	*a*	*b*	*c*	*d*	*e*	*F*
Frek.	50%	20%	10%	9%	5%	6%

Bila menggunakan 3 bit untuk mengkodekan setiap karakter (*fixed-length code*):

Tabel 2 Pengkodean huruf dengan *fixed-length code*

Kar.	*a*	*b*	*c*	*d*	*e*	*f*
Frek.	50%	20%	10%	9%	5%	6%
Kode	000	001	010	011	100	111

Bila sebuah string ‘baaf’ akan dikodekan dengan menggunakan tabel di atas akan dikodekan sebagai 001000000111 atau menggunakan 12 bit.

Dengan menggunakan *fixed-length code*, penyimpanan sebuah teks yang berisi 10.000 karakter akan memakan tempat sebanyak 30.000 bit. Jumlah tersebut bukanlah jumlah yang cukup efisien. Sementara itu, bila menggunakan prinsip pengodean Huffman, akan diperlukan jumlah bit yang lebih sedikit untuk menyimpan sebuah data.

Berikut adalah tabel yang menyajikan representasi bit untuk setiap karakter dengan menggunakan pengkodean Huffman.

Tabel 3 Pemetaan Karakter dengan Kode Prefiknya

Kar.	*a*	*b*	*c*	*d*	*e*	*f*
Frek.	50%	20%	10%	9%	5%	6%
Kode	0	101	100	111	1101	1100

Bila menggunakan tabel di atas, string ‘baaf’ akan dikodekan sebagai 101001100 atau 9 bit.

Berkas 10.000 karakter yang disebutkan sebelumnya bila dikodekan dengan prinsip Huffman akan membutuhkan tempat sebanyak:

$((0{,}5 \times 1) + (0{,}2 \times 3) + (0{,}1 \times 3) + (0{,}09 \times 3) + (0{,}05 \times 4) + (0{,}06 \times 4)) \times 10.000 = 21.100$

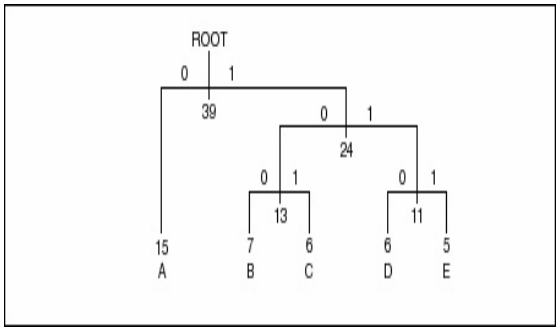
Dari hasil perhitungan di atas dapat dilihat bahwa pengkodean teks dengan menggunakan prinsip Huffman dapat mengurang jumlah bit yang diperlukan yang berarti bahwa makin sedikit pula ukuran berkas yang diperlukan.

Dari contoh pengkodean di atas dapat dilihat bahwa suatu kode itu tidak menjadi awalan bagi kode untuk karakter yang lainnya. Hal ini harus dilakukan untuk membedakan bit-bit yang akan dikodekan ulang. Prinsip seperti ini disebut juga *prefix code* atau kode prefiks.

Untuk melakukan pengkodean dengan algoritma Huffman, ada beberapa langkah yang harus dilakukan:

1. Membangun pohon biner yang akan menjadi dasar untuk melakukan susbtitusi.
2. Melakukan *parsing* terhadap berkas yang akan dikodekan untuk melakukan substitusi dengan representasi bit yang telah dihasilkan oleh pohon biner.

Untuk langkah pertama, hal yang harus dilakukan adalah membangun sebuah pohon biner yang akan menunjukkan representasi bit yang sesuai. Contoh pohon biner tersebut dapat dilihat di gambar berikut:



Gambar 2 Pohon Biner untuk Algoritma Huffman

Dalam pohon dalam gambar di atas, daun pohon melambangkan karakter yang akan dikodekan sementara bit-bit yang menjadi lintasan untuk mencapai karakter akan menjadi representasi bitnya. Untuk gambar di atas, tabel substitusinya dapat dilihat sebagai berikut:

Tabel 4 Pemetaan karaker dan representasi bit-nya

Karakter	Representasi Bit
A	0
B	100
C	101
D	110
E	111

Kode prefiks dapat di-dekode-kan kembali tanpa adanya ambiguitas (*unambiguously*). Proses dekompresi atau penirmampatan dari algoritma Huffman dilakukan melalui langkah-langkah berikut:

1. Mulai dari akar.
2. Baca sebuah bit dari string biner.
3. Untuk setaip bit pada langkah 2, lakukan traversal pada cabang yang bersesuaian.
4. Ulangi langkah 2 dan 3 sampai bertemu daun. Kodekan rangkaian bit yang telah dibaca dengan karakter daun.
5. Ulangi dari langkah 1 sampai semua bit di dalam string habis.

Contohnya bila ada sebuah string ‘1000110’ hasil kompresi dengan menggunakan kode Huffman berdasarkan gambar pohon di atas akan dapat di-dekode-kan menjadi:

100= B
0 = A
110 = D

Untuk membangun sebuah pohon biner Huffman seperti gambar di atas, dapat dilakukan hal sebagai berikut:

1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-*assign* dengan frekuensi kemunculan karakter tersebut.
2. Gabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari dua buah pohon penyusunnya.
3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon.

Agar pemilihan dua pohon yang akan digabungkan dapat berlangsung dengan cepatm semua pohon yang ada selalu terurut menaik berdasarkan frekuensinya.

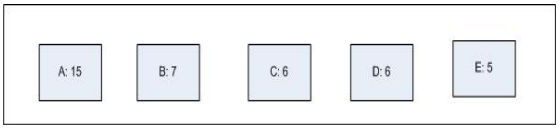
Untuk melakukan pengkodean terhadap data yang akan dikodekan, langkah yang mudah adalah dengan membangun sebuah tabel subsitusi terlebih dahulu. Tabel substitusi ini berisi pemetaan sebuah karakter dengan representasi bitnya. Setelah tabel ini telah terbentuk, proses pengkodean dapat dilakukan dengan mudah, yaitu dengan melakukan pembacaan data dan langsung mensubstitusinya dengan menggunakan tabel substitusi.

Berikut adalah contoh pembentukan pohon Huffman dari tabel frekuensi berikut:

Tabel 5 Frekuensi kemunculan huruf dalam studi kasus

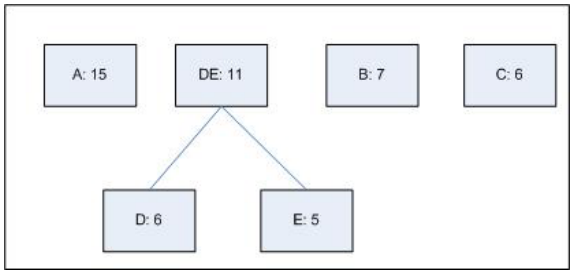
A	B	C	D	E
15	7	6	6	5

Langkah 1 :



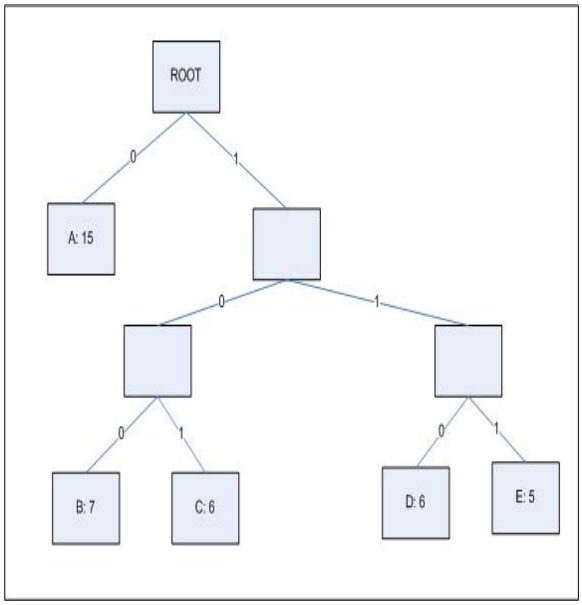
Gambar 3 Langkah I Pembentukan Pohon

Dua simpul yang memiliki frekuensi terendah adalah D dan E. Oleh karena itu dua simpul tersebut akan digabungkan sehingga akan terlihat menjadi seperti di bawah ini.



Gambar 4 Langkah II Pembentukan Pohon

Bila langkah-langkah di atas terus dilakukan, akan terbentuk sebuah pohon yang berbentuk seperti berikut:



Gambar 5 Pohon Huffman Hasil Iterasi Terakhir

Ongkos (*cost*) penggabungan dua buah pohon pada sebuah akar setara dengan jumlah frekuensi dua buah yang digabung. Operasi pembentukkan pohon di atas menggunakan strategi *Greedy* karena dalam setiap langkahnya ia memilih simpul dengan jumlah frekuensi paling kecil tanpa memikirkan bagaimana langkah-langkah selanjutnya. Algoritma Huffman sendiri memiliki kompleksitas algoritma O (n log n) untuk suatu himpunan dengan jumlah n karakter.

4. Penerapan Algoritma Huffman dalam Kriptografi

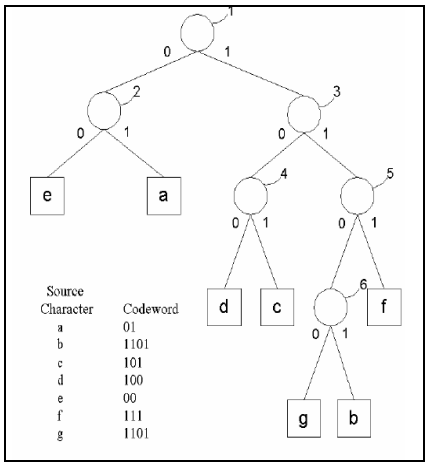
Penerapan algoritma Huffman dalam kriptografi dapat dilakukan dengan beberapa metode alternatif. Di antara cara-cara tersebut ada metode yang menghilangkan makna kompresi demi meningkatkan aspek keamanan. Akan tetapi, secara umum yang menjadi kunci untuk melakukan enkripsi dan dekripsi dalam algoritma Huffman ini adalah pohon Huffman yang dibentuk.

4.1 Penerapan dengan Kunci Berupa bit-bit pen-*shuffle*

Penerapan algoritma Huffman untuk melakukan enkripsi didapatkan dari [6]. Proses enkripsi dengan cara ini memiliki tahapan-tahapan yang mirip dengan melakukan kompresi dengan menggunakan algoritma Huffman. Perbedaannya adalah adanya *shuffling* (pengacakan) terhadap pohon Huffman. Pengacakan terhadap pohon Huffman ini dilakukan sebelum terjadinya proses pengkodean. Hasil pengacakan akan menjadi pohon Huffman yang baru yang akan digunakan dalam proses pengkodean.

Proses pengacakan pohon Huffman akan berdasarkan sebuah kunci yang dipilih. Kunci yang dipakai adalah sebuah kunci binari dengan panjang sebanyak simpul interior yang ada di dalam pohon Huffman. Setiap bit dari kunci tersebut akan menandakan apakah suatu simpul interior beranak dua yang berkoresponden perlu di-*shuffle*. Bila kunci benilai 1 simpul interior tersebut akan di-*shuffle* dan sebaliknya bila bernilai 0 tidak akan dilakukan *shuffling* terhadap simpul interor tersebut. Proses *shufflling* yang dimaksud di sini adalah melakukan pertukaran tempat antara simpul anak dari suatu simpul interior.

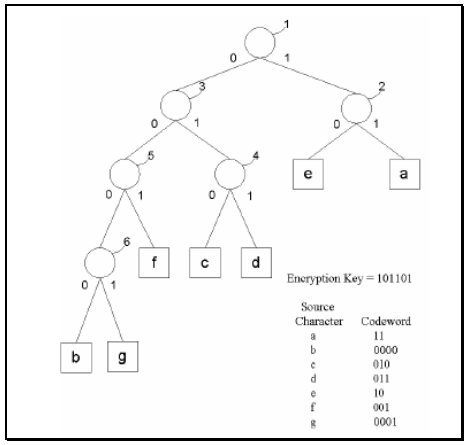
Sebagai contoh, berikut adalah suatu pohon Huffman yang telah terbentuk dari proses pembentukan pohon seperti yang diterangkan di bab sebelumnya.



Gambar 6 Pohon Huffman Asal

Untuk melakukan proses *shuffling* hal yang harus dilakukan terlebih dahulu adalah dengan memberikan nomor kepada setiap simpul interior dalam pohon Huffman. Perlu diperhatikan bahwa simpul interior yang diberi nomor hanyalah simpul interior yang memiliki dua anak. Proses penomoran sendiri dapat dilakukan dengan berbagai cara, salah satunya adalah dengan melakukan traversal terhadap pohon Huffman dan memberi nomor dari atas ke bawah, kiri ke kanan. Penomoran di gambar pohon di atas dilakukan dengan cara seperti itu.

Setelah penomoran dilakukan hal berikutnya yang dilakukan adalah melakukan *shuffling* berdasarkan kunci binari yang ada. Misalnya kunci yang akan digunakan untuk men-*shuffling* pohon di atas adalah '101101'. Dengan melakukan *shuffling* tersebut, akan terbentuklah pohon Huffman seperti berikut ini:



Gambar 7 Pohon Huffman Hasil *Shuffling*

Pohon Huffman hasil *shuffling* di atas akan digunakan sebagai pohon acuan dalam melakukan pengkodean. Patut dicatat bahwa proses *shuffling* pada pohon Huffman tidak akan membuat proses terkompresi terganggu karena yang dilakukan dalam proses *shuffling* hanyalah pertukaran tempat dalam satu simpul interior sehingga tidak akan mengubah panjang kode untuk setiap karakternya.

4.1.1 Tingkat Keamanan Enkripsi

Dari pembahasan di bab sebelumnya, dapat dilihat bahwa letak keamanan Huffman adalah di kuncinya, dalam hal ini adalah tingkat panjang kuncinya. Dalam contoh di bab sebelumnya, panjang kunci yang digunakan hanyalah 8 bit sehingga kemungkinan kunci ditebak dengan cara *exhaustive search* adalah 2^6 atau 64 kemungkinan. Angka 64 tersebut tentulah sangat kecil. Tanpa bantuan komputer pun, kunci dapat ditemukan dengan cukup cepat. Oleh karena itu, penentuan panjang kunci untuk algoritma Huffman ini sangatlah penting.

Seperti disebutkan sebelumnya, panjang kunci di algoritma enkripsi dengan Huffman ini bergantung kepada banyaknya simpul interior. Banyak simpul interior dari pohon Huffman sendiri bergantung kepada banyaknya karakter yang akan dikodekan. Bila menggunakan bahwa satu karakter adalah 1 byte sehingga terdapat kemungkinan 256 perbedaan karakter, akan didapatkan maksimum jumlah simpul interior dari pohon Huffman sebanyak 255. Dengan jumlah simpul interior sebanyak itu, akan didapatkan panjang kunci sebesar 2^{255} atau 5,7 x 10^{76}. Kemungkinan kunci sepanjang itu akan sangat sulit ditebak, bahkan dengan bantuan komputer yang sangat banyak. Dengan panjang kunci 128 bit pun dibutuhkan waktu 5,4 x 10^{18} tahun untuk mencoba kemungkinan kunci dengan menggunakan komputer yang dapat mencoba kemungkinan satu juta kunci setiap detiknya.

Akan tetapi, pada kenyataannya pada sebuah berkas tidak selalu ditemui variasi karakter sebanyak itu. Namun demikian, variasi sebanyak 64 karakter berbeda sudah cukup untuk memberikan keamanan, terutama bila hanya digunakan untuk data-data sehari-hari seperti email ataupun berkas-berkas multimedia.

Dari pembahasan di atas, dapat dilihat bahwa enkripsi dengan menggunakan algoritma Huffman ini cukup aman bila diserang dengan *exhaustive search attack* atau *brute force attack.*

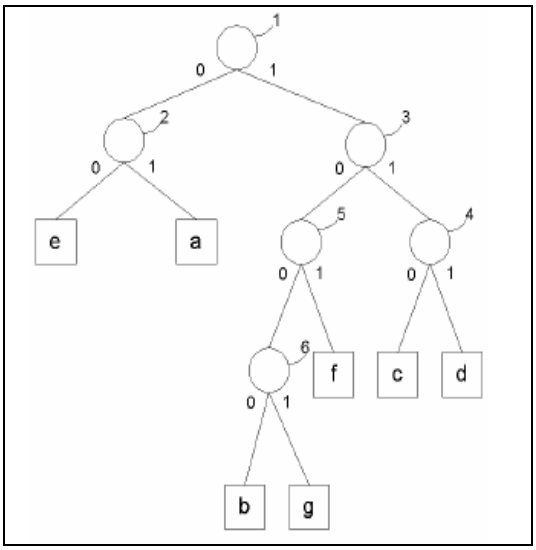
Akan tetapi, ada cara lain untuk melakukan penyerangan terhadap algoritma ini, yaitu dengan menggunakan metode *chosen-plaintext attack* (serangan yang dilakukan oleh seorang kriptanalis yang dapat memilih *plaintext* untuk dienkripsikan yang kemudian akan mengarahkannya ke penemuan kunci).

4.1.2 Peningkatan Keamanan

Seperti disebutkan di akhir bab sebelumnya, bahwa pengenkripsian dengan menggunakan teknik pengkodean Huffman dapat diserang dengan menggunakan metode *chosen-plaintext attack*. Untuk mengatasi hal tersebut, ada dua langkah yang dapat diambil.

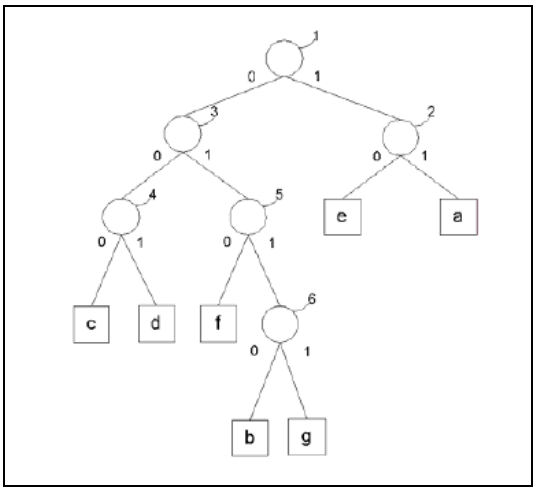
Pertama, adalah dengan menggunakan pembangkit bilangan acak semu (*pseudo random*). Bilangan yang dibangkitkan adalah sepanjang m di mana m adalah sebuah integer yang lebih panjang dari tinggi pohon atau dari panjang terbesar dari kode-kode bit yang ada. Setelah itu, dilakukan proses *exclusive or* antara bilangan yang dibangkitkan tadi dengan m bit pertama dari keluaran yang dihasilkan algoritma Huffman.

Langkah kedua yang ditambahkan adalah dengan memberikan skema pertukaran, seperti pertukaran yang ada di bab sebelumnya, hanya saja pertukaran (*swapping*) ini dilakukan setiap kali ada kode bit yang dienkripsi. Untuk melakukan hal tersebut, setiap simpul interior yang dilewati ketika menelusuri pohon Huffman dalam proses enkripsi diasosiasikan dengan kunci binari. Jadi, simpul interior pertama yang dilewati oleh suatu karakter yang dienkripsi akan diasosiasikan dengan bit pertama dari kunci dan begitu pula dengan simpul interior yang lain. Sama dengan proses *shufflling* di penjelasan sebelumnya, untuk setiap bit kunci yang bernilai 1 maka anak dari simpul interior yang berasosiasi akan dipertukarkan, dan bila bernilai nol tidak akan dipertukarkan. Misalnya kita akan mengenkripsi string ‘ab’ dengan pohon Huffman yang ada pada gambar 7. Ketika ‘a’ dikodekan, simpul-simpul interior yang dilewati adalah simpul 1 dan 2. Simpul 1 akan diasosiasikan dengan kunci 1 sementara simpul 2 akan diasosiasikan dengan kunci 0. Hal tersebut berarti anak dari simpul 1 akan dipertukarakan sementara anak dari simpul 2 tidak. Hasil pertuakaran tersebut diperlihatkan di gambar berikut ini.



Gambar 8 Pohon Huffman yang sudah diacak ketika ‘a’ dikodekan

Langkah selanjutnya adalah mengkodekan karakter ‘b’. Pengkodean karakter ini akan menggunakan pohon Huffman setelah pengkodean karakter ‘a’. Simpul-simpul yang dilewati ketika mengkodekan ‘b’ adalah 1, 3, 5, dan 6. Masing-masing simpul tersebut akan diasosiasikan dengan kunci 3, 4, 5, dan 6 atau berlanjut dari hasil pengkodean karakter ‘a’ sebelumnya. Simpul-simpul yang anaknya akan dipertukarkan adalah simpul 1, 3, dan 6 sehingga akan membentuk sebuah pohon Huffman seperti di bawah ini.



Gambar 9 Pohon Hufmman setelah ‘b’ dikodekan

4.2 Penerapan Algoritma Huffman secara Normal

Penerapan algoritma Huffman dalam kriptografi dapat juga dilakukan secara normal yaitu dengan cara yang sama dengan yang asli. Dalam hal ini yang menjadi kunci adalah pohon Huffman itu sendiri. Memang, hal ini akan menjadi kurang implementatif karena tentunya akan menyulitkan penggunanya jika sebuah kriptografi menggunakan kunci sebuah strukur pohon atau dapat pula berbentuk kamus pemetaan. Akan tetapi, ada baiknya pula bila hal ini dibahas untuk setidaknya menambah pengetahuan.

Pengkodean dengan cara ini dilakukan dengan cara yang sama dengan melakukan kompresi. Hanya saja, jika pada proses kompresi pohon Huffman yang terbentuk biasanya menjadi bagian dari berkas yang dikompresi (misalnya disimpan di header berkas), pada algoritma Huffman yang diterapkan di proses enkripsi, pohon Huffman yang dijadikan acuan untuk melakukan dekripsi akan disembunyikan. Dalam pembahasan di bab ini, penulis tidak akan mempertimbangkan masalah pengimplementasian. Masalah yang akan dibahasa hanyalah menyangkut faktor keamanannya.

Dalam pembahasan mengenai algoritma Huffman itu sendiri, dapat dilihat bahwa pembentukan pohon Huffman sangat tergantung kepada frekuensi kemunculan karakter pada suatu berkas. Dengan frekuensi kemunculan karakter yang berbeda, akan dapat menghasilkan pohon Huffman yang berbeda. Dari hal tersebut, dapat diindikasikan bahwa proses kriptanalis yang cukup efektif terhadap algoritma enkripsi dengan Huffman ini adalah dengan mengetahui distribusi frekuensi huruf yang ada. Distributsi frekuensi huruf ini disebut juga *Probability Mass Function* (PMF). Mengasumsikan bahwa seorang kriptanalis tidak mengetahui mengenai PMF ini adalah sesuatu yang kurang tepat karena PMF ini dapat ditebak terutama bila berkas yang dienkripsi adalah sebuah berkas teks. Mengapa? Karena dalam berkas-berkas teks bahasa Inggris yang ada di berbagai literatur ternyata ditemukan bahwa ada kemiripan frekuensi antara berkas-berkas tersebut. Sebagai contoh, dari kebanyakan berkas-berkas yang diuji tersebut ternyata huruf E adalah huruf yang paling sering muncul dalam literatur bahasa Inggris, diikuti oleh huruf T, A, O, I, N, S, H, dan R.

Dalam melakukan proses kriptanalisis terhadap algoritma Huffman ini, hal pertama yang harus dilakukan adalah memecahkan kode prefiks yang ada dalam *ciphertext*. Seperti sudah dibahas sebelumnya, pengkodean algoritma menggunakan kode prefiks di mana suatu kode bukan merupakan prefiks dari kode lainnya dan antara kode yang satu dengan kode yang lainnya dapat memiliki panjang yang berbeda (*variable-length code*). Untuk melakukan kriptanalisis, diperlukan pemecahan terhadap kode prefiks ini sehingga dapat diketahui kode-kode prefiks yang digunakan.

Untuk melakukan hal tersebut, cara yang dapat ditempuh adalah dengan melakukan enumerasi terhadap *ciphertext* yang dimiliki kriptanalis. Proses enumerasi secara detail tidak akan dibahas di dalam makalah ini mengingat prosesnya yang cukup rumit dan panjang. Prosees enumerasi dapat lebih jelas dan rinci dibaca di [4]. Akan tetapi, secara garis besar algoritma untuk melakukan enumerasi terhadap kode prefiks adalah sebagai berikut:

1. Cari semua kemungkinan himpunan kode.
2. Untuk sekuens S_0 dalam *ciphertext* cek semua kandidat himpunan kode dengan menggunakan algoritma pengecekan apakah suatu himpunan kode bisa digunakan untuk melakukan pengkodean atau tidak.
3. Untuk himpunan kode yang dapat digunakan untuk melakukan proses pengkodean, simpan himpunan tersebut dan bagi yang tidak dapat digunakan dibuang.
4. Jika pada proses tiga himpunan kode yang tersisa lebih dari satu, lakukan kembali langkah 2 untuk sekuens *ciphertext* yang lain.

Setelah kode prefiks dapat dipecahkan dan PMF yang ada telah dapat ditebak, maka pemecahan algoritma Huffman menjadi mudah. Untuk menghindari terjadinya hal ini, beberapa peningkatan keamanan diimplementasikan ke dalam algoritma Huffman ini, salah satunya adalah dengan menambahkan beberapa kode bit acak ke dalam kode prefiks Huffman. Dengan melakukan hal tersebut, didapatkan bahwa proses pemecahan kode akan menjadi *NP-complete*. Akan tetapi, cara ini memiliki kelemahan karena akan mengurangi tingkat kompresi yang dilakukan oleh algoritma Huffman. Cara lain yang diajukan adalah dengan cara meng-XOR-kan sekuens bit yang telah terkompresi dengan sekuens bit yang didapatkan

dari mengenkripsikan sebuah bilangan umpan (*seed*). Dengan cara tersebut, tingkat kompresi dapat dijaga sekaligus juga meningkatkan keamanan dari algoritma Huffman untuk melakukan enkripsi.

5. Perbandingan dengan Algoritma Kompresi Lainnya

Seperti disebutkan di bab pendahuluan, bahwa selain Huffman, beberapa algoritma kompresi data lain juga telah dicoba untuk diimplementasikan dalam bidang kriptografi. Di bab ini akan dicoba diadakan pembahasan singkat mengenai aplikasi algoritma-algoritma tersebut dalam dunia kriptografi.

5.1 Algoritma Shannon-Fano

Algoritma Shannon-Fano merupakan algoritma kompresi data yang sudah cukup tua. Dibandingkan dengan algoritma Huffman, tingkat kompresi data algoritma Shannon-Fano ini kurang baik. Akan tetapi dari segi kriptografi, memiliki keamanan yang cukup baik [4].

5.2 Algoritma Lampel-Ziv

Penerapan algoritma Lampel-Ziv dalam kriptografi salah satunya dilakukan dengan *pseudo random shuffle* seperti yang terjadi pada algoritma Huffman pada bab 4.1. Dengan menggunakan teknik tersebut, didapati bahwa teknik ini dapat diaplikasikan dengan cukup baik karena tidak seperti algoritma Huffman yang mengandalkan panjang kunci untuk melakukan pengamaman, algoritma enkripsi dengan Lampel-Ziv ini mengandalkan pada umpan (*seed*) pembangkit bilangan acak semu yang meupakan variabel internal dalam proses pengkodean.

5.3 Algoritma *Arithmetic Coding*

Algoritma *Arithmetic Coding* ini merupakan algoritma yang dikembangkan sebagai pengganti algoritma Huffman. Pengaplikasiannya dalam bidang kriptografi sama dengan pengaplikasian algoritma Lampel-Ziv, yaitu sama-sama mengandalkan pada umpan pembangkit bilangan semu.

6. Kesimpulan

Penerapan Algoritma Huffman dalam kriptografi cukup layak bila menggunakan alternatif penerapan yang ada di bab 4.1 karena pertimbangan dari segi keamanan yang cukup mumpuni dan dari segi kunci yang cukup praktis. Sementara itu, penerapan algoritma Huffman yang dibahas di bab 4.2 selain penerapannya yang kurang praktis karena bentuk kuncinya yang panjang, penerapan dengan cara tersebut ternyata tidak dapat memberikan proteksi yang cukup. Proteksi yang cukup sebenarnya dapat diberikan dengan cara melakukan berbagai peningkatan (*enhancement*) terhadap algoritma Huffman. Selain itu, dapat disimpulkan juga bahwa selain algoritma Huffman, ada pula algoritma-algoritma kompresi data lainnya yang dapat dimanfaatkan dalam dunia kriptografi. Secara umum, algoritma-algoritma tersebut cukup layak untuk diaplikasikan dalam pengenkripsian data.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2005. *Diktat Kuliah IF2251 Strategi Algoritmik*. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[2] Munir, Rinaldi. 2005. *Diktat Kuliah IF5054 Kriptografi*. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[3] Nelson, Mark. *The Data Compression Book*. Forge's E books.

[4] Ruan, Xiaoyu. 2005. *Using an Innovative Coding Algorithm for Data Encryption*. http://www.ece.ndsu.nodak.edu/~katti/pdf/jp01.pdf. Tanggal akses: 22 September 2006.

[5] Wagner, Neal. 2003. *The Laws of Cryptography with Java Code*. http://www.cs.utsa.edu/~wagner/lawsbook/. Tanggal akses: 8 Oktober 2006.

[6] Wang, Chung-E. 2006. *Cryptography in Data Compression*. Code Breaker's Journal March 2006.