Teknik *secret sharing* yang efektif pada berkas yang terkompresi dengan menggunakan Algoritma Huffman

Ibnul Qoyyim [1)]

1) Jurusan Teknik Informatika ITB, Bandung, email: if14066@students.if.itb.ac.id

Abstract – *Makalah ini membahas tentang secret sharing yang efektif pada berkas yang dikompresi dengan algoritma huffman.*

Kata Kunci: *secret sharing, Huffman hoding.*

1. PENDAHULUAN

Secret sharing adalah metode untuk mendistribusikan rahasia kepada sebuah grup partisipan yang berjumlah n, setiap partisipan diberikan *share* dari rahasia, rahasia hanya dapat disusun jika k *share* digabungkan dengan k <= n, setiap *share* atau gabungan *share* yang berjumlah kurang dari k tidak berguna untuk mengetahui rahasia. [2]. Rahasia yang dibagi adalah berkas yang dimampatkan menggunakan Huffman.

Huffman adalah cara memampatkan data dengan mengkodekan setiap karakter dalam data dengan kode yang lebih pendek. Untuk meminimumkan jumlah bit yang dibutuhkan, panjang kode untuk setiap karakter sedapat mungkin diperpendek, terutama untuk karakter yang kekerapannya (*frequency*) kemunculannya besar[1].

2. DASAR TEORI

Secret sharing pada berkas yang dikompresi menggunakan Huffman dapat dibuat lebih efektif dibanding berkas umum karena untuk menirmampatkan berkas diperlukan pohon Huffman atau tabel Huffman sehingga pohon atau tabel Huffman dapat dianalogikan sebagai kunci untuk membuka berkas Huffman.

Shamir secret sharing cukup baik digunakan untuk menshare data umum, namun kurang efektif secara ukuran karena ukuran share sama dengan atau lebih besar dari ukuran rahasia sehingga keseluruhan share yang diperlukan memiliki ukuran sekitar k kali ukuran sumber, selain itu *shamir secret sharing* tidak bisa memanfaatkan sifat dari data khusus seperti graf, pohon dsb. Oleh karena itu penulis berusaha mencari skema *secret sharing* yang efektif digunakan pada header berkas Huffman agar ukuran *share* dapat seminimal mungkin.

Naive secret sharing secara ukuran cukup baik namun kurang aman untuk digunakan untuk menshare data umum karena share yang terkumpul meskipun kurang dari k memberi informasi signifikan terhadap rahasia [5], namun pada masalah Huffman asalkan header Huffman aman, body berkas Huffman dapat menggunakan *naive secret sharing* karena informasi body tanpa header Huffman tidak memiliki makna yang berarti.

3. HASIL DAN PEMBAHASAN

Representasi header Huffman bisa berupa Huffman *table* atau Huffman *tree* oleh karena itu metode secret sharing yang dapat digunakan pada header Huffman dapat bermacam macam. misal mengencode dengan cara biasa kemudian byte-byte header di*secret sharing* menggunakan Shamir secret sharing, atau merepresentasikan Huffman *tree* (graf) sebagai matriks kemudian melakukan *secret sharing* khusus matriks atau menggunakan variasi dari prinsip visual kriptografi dimana pixel-pixel digantikan oleh bit-bit.

Variasi dari prinsip visual kriptografi pada matriks Huffman sebuah elemen matriks dengan value [0,1] dibagi menjadi beberapa sub elemen dengan masing masing sub elemen memiliki value [0,1] untuk mengetahui apakah value sebenarnya dari element tersebut didapatkan dengan mengoperasikan sub elemen setiap matriks dengan operasi OR, *Hamming Weight* dari elemen matriks tsb adalah ∑ hasil operasi OR sub elemen yang bernilai 1 kemudian membandingkannnya dengan tabel berikut :

Value	H(V)
0	$H(V) \geq d$
1	$H(V) < d - \alpha m$

Dengan :
H(V) : Hamming Weight = ∑ subelemen hasil operasi OR subelemen dari tiap share
d : ambang batas ($1 \leq d \leq m$)
α : kontras
m : jumlah subelemen

modifikasi pada visual kriptografi adalah perubahan sub pixel menjadi bit bit pada matriks.

Untuk enkripsi mula-mula buat Huffman tree kemudian representasikan graf tersebut menjadi matriks. untuk pembagiannya sama dengan visual kriptografi biasa yaitu membuat dua buah koleksi matriks boolean berukuran n x m, C0 dan C1. Untuk membagi elemen 0 pada matriks, dipilih salah satu dari matriks pada C0, sedangkan untuk membagi

elemen 1, dipilih salah satu matriks dari C1. Matriks yang terpilih merupakan representasi dari m subpixel pada masing-masing n lembar transparan. Solusi dianggap valid jika seluruh kondisi berikut terpenuhi:

1. Untuk sembarang matriks M pada C0, hasil operasi OR dengan V dengan sembarang baris k dari n memenuhi H(V) < d - am.
2. Untuk sembarang matriks M pada C1, hasil operasi OR dengan V dengan sembarang baris k dari n memenuhi H(V) = d.
3. Untuk sembarang j < k baris yang dipilih, submatriksnya muncul dengan frekuensi yang sama pada C0 dan C1.

Sementara untuk dekripsinya mula-mula untuk setiap sub elemen pada suatu share OR-kan dengan sub elemen pada share yang lain, matriks hasil dari operasi OR tersebut ditentukan H(V) setiap elemennya yaitu ∑ subelemen dalam satu elemen. Kemudian ditentukan apakah elemen tersebut bernilai 1 atau 0 dari tabel H(V) diatas matriks hasilnya ditransformasi menjadi huffman tree yang akan digunakan untuk men decode body.

seperti yang dijelaskan diawal secret sharing pada body dapat menggunakan *naive secret sharing* namun harus dicari cara agar skema k to n dapat terlaksana yaitu untuk mendapatkan keseluruhan rahasia cukup dengan mengumpulkan k share dari n share yang dibuat, oleh karena itu *naive secret sharing* yang dipakai memiliki redundansi,

Untuk mendapatkan skema pembagian *naive secret sharing* per sharenya dapat diperoleh beberapa cara:

1. Brute Force

   Pendekatan secara brute force diperoleh dengan cara mengkombinasikan m bagian body kepada n share dan mengecek apakah semua kemungkinan k share dapat membentuk keseluruhan rahasia.

   Algoritma secara garis besar:

```
do
{
   for i = 1 to n
   {
      share[ i ] = kombinasi_m_bagian();
   }


   for a = 1 to n
   {
       for b = 1 to n
       {
      … hingga kedalaman k…
              test     =     test     and
is_complete(share[  a  ],  share[  b  ]  …
share[ k ])
      }
   }
}
while test = false
```

   Dari algoritma diatas dapat dilihat bahwa bottle neck terdapat pada bagian test sehingga kompleksitas algoritma dalam notasi O adalah O (n^k).

2. Pencarian skema yang lebih baik

   2.1. Skema yang pasti

   Skema dibawah ini pasti dan karenanya dipakai basis jika terdapat algoritma yang rekursif

   k = 1 maka seluruh share memiliki elemen yang sama, misal :

```
Untuk k = 1, n = 3 masing masing share adalah :
a
a
a
```

   k = n maka tidak ada rendundansi seluruh share memiliki elemen yang berbeda-beda, misal:

```
Untuk k = 2, n = 2 masing masing share adalah :
a
b
```

   2.2. Skema lainnya

   Untuk skema naïve lainnya digunakan percobaan sebagai berikut untuk mendapatkan algoritma:

Dari hasil hipotesis :

```
Untuk k = 2, n = 3 masing masing share adalah :
ab
bc
ac
```

```
Sementara untuk k = 3, n = 4 masing masing
share adalah :
abd
bce
acf
def
```

   Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k+1 to n+1 turunannya dengan cara :

   Buat skema untuk k+1 to n elemen

   Untuk setiap share asal ditambah share baru yang

berbeda satu dengan lain
Buat suatu share yang merupakan gabungan share share yang ditambahkan pada share asal

Dari hasil hipotesis :

Untuk k = 2, n = 3 masing masing share adalah :
ab
bc
ac
Sementara untuk k = 1, n = 2 masing masing share adalah :
b
b

Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k-1 to n-1 turunannya dengan cara :
Pilih salah satu share yang akan dieliminasi
Untuk setiap share lainnya kurangkan elemen yang terdapat pada share yang akan dieliminasi
Eliminasi share yang terpilih

Dari hasil hipotesis :

Untuk k = 2, n = 3 masing masing share adalah :
ab
bc
ac
Sementara untuk k = 2, n = 2 masing masing share adalah :
a
c

Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k to n-1 dengan ($k \leq (n-1)$) turunannya dengan cara :
Pilih salah satu share yang akan dieliminasi
Untuk setiap sahare selain yang terpilih gunakan operasi AND dengan share yang terpilih
Eliminasi share yang terpilih

Dari hasil hipotesis :

Untuk k = 2, n = 3 masing masing share adalah :
ab
bc
ac
Sementara untuk k = 2, n = 4 masing masing share adalah :

abd
bcd
acd
abc

Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k to n+1 turunannya dengan cara :
Buat suatu share baru yang merupakan union seluruh share yang lain.
Buat skema k-1 to n bagikan ke share selain yang baru saja dibuat

Dari hasil hipotesis :

Untuk k = 3, n = 4 masing masing share adalah :
abd
bce
acf
def
Sementara untuk k = 2, n = 4 masing masing share adalah :
abd
bcd
acd
abc

Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k-1 to n dengan ((k-1) > 1) turunannya dengan cara :
Pilih salah satu share yang akan diganti
Untuk setiap share lainnya kurangkan elemen yang terdapat pada share yang akan diganti
Ganti share yang terpilih dengan union seluruh share yang lain.
Buat skema k-2 to n bagikan ke share selain yang baru saja diganti

Dari hasil hipotesis :

Untuk k = 2, n = 4 masing masing share adalah :
abd
bcd
acd
abc
Sementara untuk k = 3, n = 4 masing masing share adalah :
abd

bce

acf

def

Dari hasil diatas dapat disimpulkan bahwa untuk setiap skema k to n yang terdefinisi dapat dibuat skema k+1 to n dengan ((k+1) < n) turunannya dengan cara :
Pilih salah satu share yang akan diganti
Untuk setiap share lainnya kurangkan elemen yang yang beririsan di semua share selain yang terpilih
Buat skema k+1 to n-1 bagikan ke share selain yang baru saja diganti
Ganti share yang terpilih dengan gabungan share share yang ditambahkan pada share asal

Dari hasil hipotesis diatas dapat dirumuskan algoritma transformasi sebagai berikut :

```
increment k and n (array of share_scheme)
→ array of share_scheme
{
   new_share_scheme = create_scheme (k+1,n)
   for i = 1 to count(share_scheme)
   {
      new_share_scheme[ i ] = share_scheme[
      i ] + new_share_scheme[ i ]
      union = union + share_scheme[ i ]
   }
   new_share_scheme[ i + 1 ] = union
   →new_share_scheme
}
```

```
decrement k and n (array of share_scheme)
→ array of share_scheme
{
   chosen_share_scheme                    =
   share_scheme[count(share_scheme)]
   for i = 1 to count(share_scheme) - 1
   {
      new_share_scheme[ i ] = share_scheme[
      i ] - chosen_share_scheme
   }
   →new_share_scheme
}
```

```
increment n (array of share_scheme) →
array of share_scheme
{
   new_share_scheme = create_scheme (k+1,n)
   for i = 1 to count(share_scheme)
   {
      new_share_scheme[ i ] = share_scheme[
      i ] + new_share_scheme[ i ]
      union = union + share_scheme[ i ]
   }
   new_share_scheme[ i + 1 ] = union
   →new_share_scheme
}
```

```
decrement n (array of share_scheme) →
array of share_scheme
{
   chosen_share_scheme                    =
   share_scheme[count(share_scheme)]
   for i = 1 to count(share_scheme) - 1
   {
      new_share_scheme[ i ] = share_scheme[
      i ] - chosen_share_scheme
   }
   →new_share_scheme
}
```

```
increment k (array of share_scheme) →
array of share_scheme
{
   chosen_share_scheme                    =
   share_scheme[count(share_scheme)]

   new_share_scheme = create_scheme (k+1,n-
   1)

   intersect = share_scheme[ 1 ]
   for i = 1 to count(share_scheme) - 1
   {
      union = union + new_share_scheme[ i ]
      intersect    =    intersect    AND
      share_scheme[ i ]
   }

   for i = 1 to count(share_scheme) - 1
   {
      new_share_scheme[ i ] = (share_scheme[
      i ] - intersect) + new_share_scheme[ i
      ]
   }
   new_share_scheme[ i + 1 ] = union
   →new_share_scheme
}
```

```
decrement k (array of share_scheme) →
array of share_scheme
{
   chosen_share_scheme                    =
   share_scheme[count(share_scheme)]

   new_share_scheme = create_scheme (k-2,n)

   for i = 1 to count(share_scheme) - 1
   {
      union = union + share_scheme[ i ]
      new_share_scheme[ i ] = (share_scheme[
```

```
        i ] - chosen_share_scheme) +
        new_share_scheme[ i ]
    }
    new_share_scheme[ i + 1 ] = union
    →new_share_scheme
}
```

Dari algoritma diatas dapat dilihat beberapa merupakan fungsi rekursif. Fungsi tersebut dijalankan hingga mencapai basisnya yaitu k = 1 atau k = n. namun untuk mencapai keadaan itu bisa melalui berbagai macam cara melihat cukup banyak algoritma yang dapat digunakan, karena itu agar pencarian skema berjalan dengan efektif maka perlu dibuat cara mencari pilihan terpendek (dilihat dari kedalaman rekursif fungsi), cara yang saya usulkan adalah membuat peta k/n seperti berikut :

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●							
3			●						
4				●					
5					●				
6						●			
7							●		
8								●	
9									●

● = basis k = n
○ = basis k = 1

Dari peta k/n diatas dapat dibuat rute perpendek dari basis menggunakan algoritma yang telah ditemukan, namun tidak seluruh algoritma dipakai sebaiknya algoritma yang tidak menggunakan fungsi create_scheme agar lebih efektif. Dari daftar diatas algoritma yang tidak memakai fungsi create_scheme adalah decrement k and n dan decrement n. decrement k and n membuat posisi pada peta k/n bergerak ke kiri atas sementara decrement n membuat posisi pada peta k/n menjadi kekanan, dilihat dari posisi basis dan perubahan posisi maka kedua algoritma ini tidak cukup untuk mendapatkan seluruh skema k to n yang ada karena itu diperlukan algoritma lain.
Dari 4 algoritma yang kurang efektif karena menggunakan fungsi create_scheme dua diantaranya menggunakan k+1 to n sementara yang lainnya adalah k+1 to n-1 dan k-2 to n dari peta k/n :

k\n	1	2	3	4	5	6	7	8	9
1	●○	○A	○B	○	○	○	○	○	○
2		●	A	B					
3			●	AC	BC	C	C	C	C
4				●	A	B			
5					●	A	B		
6						●	A	B	
7							●	A	B
8								●	A
9									●

● = basis k = n

○ = basis k = 1

A = daerah dimana k+1 to n merupakan basis

B = daerah dimana k+1 to n-1 merupakan basis

C = daerah dimana k-2 to n merupakan basis

Dari keempat algoritma tersebut perlu dipilih algoritma mana yang paling efektif dilihat dari berapa banyak skema yang perlu diekspansi hingga menemukan skema hasil, disini penulis mencari skema 4 to 7 angka tersebut digunakan karena letak pada peta k\n sama sama memiliki jarak 3 petak dari basis k = 1 maupun k = n.
Bila kita hanya menggunakan increment k and n

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●	x	x	x				
3			●	x	x	x			
4				●	x	x	+		
5					●				
6						●			
7							●		
8								●	
9									●

Maka skema yang perlu diekspansi = 9.

Bila kita hanya menggunakan increment n

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●							
3			●						
4				●	x?	x	+		
5					●	x?			

6						●			
7							●		
8								●	
9									●

Maka skema yang perlu diekspansi = 4 jika skema 4 to 5 dan 5 to 6 diketahui.

Bila kita hanya menggunakan increment k

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●	x	x	x	x	x		
3			●	x	x	x	x		
4				●	x	x	+		
5					●				
6						●			
7							●		
8								●	
9									●

Maka skema yang perlu diekspansi = 12.

Bila kita hanya menggunakan decrement k

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●							
3			●				x?		
4				●			+		
5					●		x		
6						●	x		
7							●		
8								●	
9									●

Seperti yang dapat dilihat diatas tidak mungkin mendapatkan skema dengan hanya menggunakan fungsi decrement k

Dari percobaan diatas dapat dilihat bahwa cara yang hanya menggunakan fungsi increment n mengekspansi paling minimal, namun fungsi tersebut masih memerlukan bantuan agar bisa terdefinisi :

Dari algoritma increment n dapat dilihat bahwa untuk mendapatkan skema k to n maka kita perlu mengetahui skema k to n-1 dan k-1 to n, digambarkan dalam peta k/n sebagai berikut :

x	+	

x		

Dari algoritma diketahui bahwa algoritma increment k and n dapat digunakan sebagai komplemen increment n, jika pencarian skema menggunakan algoritma decrement k and n ruang pencarian akan menjadi :

k\n	1	2	3	4	5	6	7	8	9
1	●○	○	○	○	○	○	○	○	○
2		●	x						
3			●	x					
4				●	x	x	+		
5					●	x			
6						●			
7							●		
8								●	
9									●

Keterangan :

+ : skema yang dicari

x : skema yang didapat menggunakan increment k

x : skema yang didapat menggunakan increment k and n

hasil : skema yang perlu diekspansi = 4

dengan cara diatas dapat dibuat algoritma create_scheme yang menggunakan increment_n dan decrement_k_and_n sebagai berikut :

```
create_scheme(int k, int n) → array of share_scheme
{
   //basis
   if (k=1)
   {
      new_element = new_elements()

      for i = 1 to n
      {
         new_share_scheme[ i ] = new_element
      }
   }
   else if (k=n)
   {
      for i = 1 to n
      {
         new_share_scheme[ i ] = new_elements()
      }
```

```
    }
    //rekurens (algoritma increment n)
    else if (n-k > 1)
    {
       share_scheme = create_scheme (k,n-1)
       new_share_scheme    =    create_scheme
       (k+1,n-1)
       for i = 1 to count(share_scheme)
       {
          new_share_scheme[     i     ]     =
          share_scheme[      i      ]      +
          new_share_scheme[ i ]
          union = union + share_scheme[ i ]
       }
       new_share_scheme[ i + 1 ] = union
       →new_share_scheme
    }
    //rekurens (algoritma increment k and n)
    else //n-k == 1
    {
       share_scheme = create_scheme (k-1,n-1)
       new_share_scheme = create_scheme (k,n-
       1)
       for i = 1 to count(share_scheme)
       {
          new_share_scheme[     i     ]     =
          share_scheme[      i      ]      +
          new_share_scheme[ i ]
          union = union + share_scheme[ i ]
       }
       new_share_scheme[ i + 1 ] = union
    }
    →new_share_scheme
}
```

Perbandingan ukuran setiap share (dalam % ukuran awal):

k\n	1	2	3	4	5	6
1	100	100	100	100	100	100
2		50	66	75	80	83
3			33	50	60	66
4				25	40	50
5					20	33
6						16

Perbandingan ukuran keseluruhan share (dalam % ukuran awal):

k\n	1	2	3	4	5	6
1	100	200	300	400	500	600
2		100	133	150	160	166
3			100	150	180	200
4				100	160	200
5					100	166
6						100

Dari hasil diatas dapat dilihat bahwa ukuran setiap share berbanding lurus dengan n dan berbanding terbalik dengan k, sementara pada ukuran keseluruhan share walaupun ukuran terbesar digunakan oleh k = 1 dan ukuran terkecil oleh k=n ukuran keseluruhan tidak berbanding lurus namun memiliki puncak di k = 0,5 n dan menurun di k < 0,5 n maupun k > 0,5 n.

4. KESIMPULAN

Dari pembahasan diatas dapat disimpulkan bahwa *secret sharing* pada berkas yang dimampatkan menggunakan Huffman coding dimungkinkan dengan cara memisahkan header dari body kemudian membagi dengan algoritma *secret sharing* yang berbeda, *secret sharing* yang *secure* seperti yang dicontohkan diatas adalah modifikasi dari visual kriptografi untuk header dan *naïve secret sharing* untuk body.

Metode yang sama diharapkan dapat digunakan untuk berkas kompresi yang lain yang memiliki header dan body. Dengan menggunakan prinsip yang sama yaitu menggunakan *secret sharing* yang *secure* untuk header dan *naïve secret sharing* untuk body.

Beberapa saran utuk pengembangan dimasa depan adalah :

1. Implementasi pada format kompresi yang lebih rumit seperti zip, rar, 7z dll.
2. Algoritma transformasi yang lebih efektif.
3. Implementasi algoritma *secret sharing* lain untuk header.

DAFTAR REFERENSI

[1] Munir, Rinaldi., Diktat Kuliah IF5054 Kriptografi, 2004

[2] Munir, Rinaldi., Slide Kuliah IF5054 Kriptografi : Skema Pembagian Data Rahasia, 2007

[3] Munir, Rinaldi., Slide Kuliah IF5054 Kriptografi : Kriptografi Visual, 2007

[4] Makalah IF5054 : Pemanfaatan Steganografi dalam Kriptografi Visual

[5] http://en.wikipedia.org diunduh 11 Desember 2007