Blok Cipher JUMT

Mario Tressa Juzar (13512016)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
mariotj.tj@gmail.com

***Abstract* — Dalam makalah ini penulis menyajikan Blok Cipher JUMT sebagai varian baru dari cipher blok yang merupakan modifikasi dan pengembangan dari blok cipher yang sudah ada yakni blok cipher JAFT. Blok cipher JUMT menggunakan kunci 128-bit dan potongan blok juga 128-bit. Pada blok cipher JUMT terdapat shiftCells, addRoundKey, subtituteSBox, shiftRow, xorOperation, dan diakhiri dengan jaringan Feistel. Untuk S-Box dari blok cipher JUMT dibuat secara acak berdasarkan kunci yang diberikan dan juga langsung membangun round key dari awal.**

Keywords — JUMT, pembangunan S-Box, pembangunan round key.

I. PENDAHULUAN

Cipher blok merupakan algoritma kriptografi yang cukup banyak digunakan dan cukup mudah untuk dibuat dan dikembangkan dewasa ini. Algoritma cipher blok berdasarkan pada plainteks yang dijadikan blok terlebih dahulu begitupun juga dengan kunci yang digunakan juga dijadikan blok terlebih dahulu. Blok yang digunakan bervariasi mulai dari 64-bit, 128-bit, 192-bit dan lainnya yang juga bisa lebih besar lagi.

Blok cipher yang dikembangkan pun juga semakin beragam tidak bergantung pada pakem yang ada. Blok cipher terus menemukan hal-hal baru untuk dimodifikasi dan untuk dikembangkan. Seiring dengan berkembangnya algoritma blok cipher maka ilmu kriptanalisis juga akan semakin berkembang. Kriptanalisis merupakan sebuah studi untuk memecahkan sebuah metoda kriptografi. Dimana ada algoritma kriptografi baru biasanya disitu akan ada orang yang berusaha untuk menemukan kelemahan dan memecahkannya.

Karena itu kita tidak hanya bisa berpegang pada satu jenis algoritma blok cipher saja. Cepat atau lambat algoritma blok cipher tersebut akan bisa dikriptanalisis/dipecahkan atau paling tidak ditemukan kelemahannya oleh orang lain. Sebuah kelemahan yang merupakan pertanda bahwa algoritma blok cipher sudah tidak aman lagi.

Atas dasar hal itu lah penulis mencoba memodifikasi dan mengembangkan blok cipher yang sudah ada sebelumnya dengan mengembangkan sebuah blok cipher baru yang diberi nama blok cipher JUMT. Blok cipher JUMT merupakan modifikasi dan pengembangan dari algoritma blok cipher yang sudah dibuat sebelumnya yakni algoritma blok cipher JAFT.

Pada blok cipher JUMT hampir mengadopsi semua teknik yang ada pada blok cipher JAFT, namun ada beberapa modifikasi dan pengembangan untuk peningkatan keamanan pada blok cipher ini. Blok cipher JUMT mengikuti sebagian besar cara pada blok cipher JAFT, namun ada beberapa perbedaan seperti pada S-Box dan pembangkitan *round key*. S-Box pada blok cipher JUMT dibangkitkan secara acak sesuai dengan kunci yang diberikan, jadi terdapat fungsi pseudo-random yang mengacak isi S-Box, sehingga untuk setiap kunci yang berbeda juga akan terdapat S-Box yang berbeda karena S-Box dibangkitkan secara acak berdasarkan kunci yang diberikan. Setelah didapatkan S-Box yang baru secara acak maka *round key* juga langsung dibangkitkan dari S-Box yang sudah ada sehingga akan terdapat 7 *round key*. *Round key* inilah yang kemudian digunakan pada putaran dalam proses enkripsi dan dekripsi dari blok cipher JUMT. Untuk selebihnya proses yang digunakan tidak jauh berbeda dari blok cipher JAFT, namun ada beberapa perubahan urutan dalam proses enkripsi dan dekripsi.

Dengan dihadirkan algoritma blok cipher yang baru ini diharapkan membuat proses pengamanan pesan menggunakan algoritma blok cipher menjadi lebih terjaga. Blok cipher JUMT diharapkan bisa menjadi lebih tahan terhadap serang dibandingkan blok cipher JAFT yang sudah ada sebelumnya.

II. DASAR TEORI

A. *Polyalphabetic Substitution Cipher*

Polyalphabetic Substitution Cipher atau cipher subtitusi abjad-majemuk menggunakan prinsip setiap huruf menggunakan kunci berbeda.

Misalkan plainteks:

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

Maka cipherteksnya akan menjadi seperti berikut:

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

B. Chiper Blok

Algoritma cipher blok merupakan salah satu dari algoritma kriptografi modern. Algoritma ini beroperasi dalam mode bit. Kunci, plainteks, dan cipherteks diproses dalam rangkaian bit. Algoritma ini tetap menggunakan gagasan pada algoritma kriptografi klasik seperti subtitusi dan transposisi, tetapi lebih rumit dan sangat sulit untuk dipecahkan.

Prinsipnya adalah sebagai berikut:

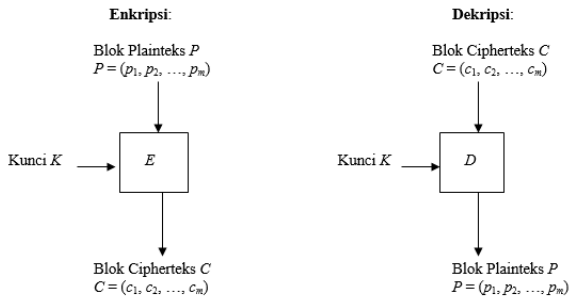
- Pesan dalam rangkaian bit dipecah menjadi beberapa blok
- *Padding bits* : merupakan bit-bit tambahan jika ukuran blok terakhir tidak mencukupi panjang blok. Padding bits mengakibatkan ukuran plainteks hasil deskripsi sedikit lebih besar dari plainteks semula.
- Pesan juga dapat dinyatakan dalam kode heksadesimal
- Bit-bit plainteks dibagi menjadi blok dengan panjang yang sama
- Panjang kunci enkripsi = panjang blok
- Enkripsi dilakukan terhadap blok bit plainteks menggunakan bit-bit kunci
- Algoritma enkripsi menghasilkan bok cipherteks yang panjangnya sama dengan panjang blok plainteks.

Blok plainteks berukuran *m* bit:

$P = (p_1, p_2, \ldots, p_m), \qquad p_i \in \{0, 1\}$

Blok cipherteks (C) berukuran *m* bit:

$C = (c_1, c_2, \ldots, c_m), \qquad c_i \in \{0, 1\}$



Gambar 1 : proses enkripsi dan dekripsi cipher blok

C. Operasi XOR

Operasi xor merupakan operasi pada level bit. Notasi yang digunakan untuk operasi xor adalah ⊕.

Operasi yang berlaku pada operasi xor:

$0 \oplus 0 = 0 \qquad 0 \oplus 1 = 1$

$1 \oplus 0 = 1 \qquad 1 \oplus 1 = 0$

Hukum-hukum yang terkait dengan operator xor:

(i) $a \oplus a = 0$

(ii) $a \oplus b = b \oplus a$

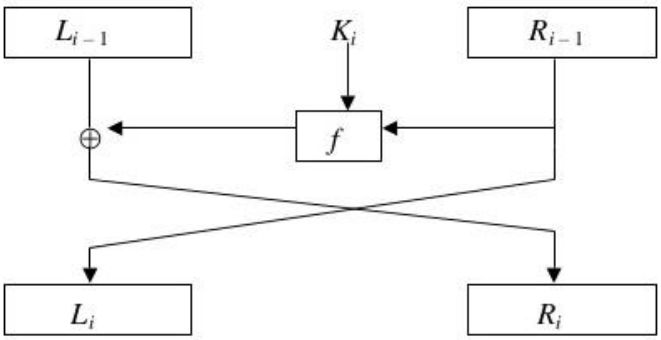
(iii) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$

Operasi xor pada level bit : jika dua rangkaian dioperasikan dengan xor, maka operasinya dilakukan dengan meng-xor-kan setiap bit yang berkoresponden dari kedua rangkain bit tersebut.

Contoh: 10011 ⊕ 11001 = 01010

D. Jaringan Feistel

Jaringan Fiestel beroperasi terhadap panjang blok data tetap sepanjang n (genap), kemudian membagi 2 blok tersebut dengan panjang masing-masing n/2, yang dinotasikan dengan L dan R. Fiestel cipher menerapkan metode cipher berulang dengan masukan pada putaran ke-i yang didapat dari keluaran sebelumnya.



Gambar 2 : struktur jaringan feistel

Secara matematis dapat dinyatakan sebagai berikut:

$L_i = R_{i-1}$

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

Ki adalah kunci untuk putaran ke-I dan f adalah fungsi transformasi.

Blok plainteks adalah gabungan L dan R awal atau secara formal plainteks dinyatakan dengan (L_0, R_0). Sedangkan blok cipherteks didapatkan dari L dan R hasil putaran terakhir setelah terlebih dahulu dipertukarkan atau secara formal cipherteks dinyatakan dengan (R_r, L_r).

Jaringan Feistel banyak dipakai pada algoritma kriptografi DES, LOKI, GOST, FEAL, Lucifer, Blowfish, dan lain-lain karena model ini bersifat reversible untuk proses enkripsi dan dekripsi.Sifat reversible ini membuat kita tidak perlu membuat algoritma baru untuk mendekripsi cipherteks menjadi plainteks.

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

Sifat reversible tidak bergantung pada fungsi f sehingga fungsi f dapat dibuat serumit mungkin.

E. Pembangkitan Bilangan Acak

Bilangan acak merupakan bilangan yang tidak dapat diprediksi kemunculannya. Namun, tidak ada komputasi yang benar-benar menghasilkan deret bilangan acak secara sempurna. Bilangan acak yang dihasilkan dengan rumus-rumus matematika adalah bilangan acak semu (*pseudo*), karena pembangkitan bilangannya dapat diulang kembali. Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (*PRNG*).

Salah satu algoritma standar dalam pembangkitan bilangan acak yakni pembangkit bilangan acak kongruen-lanjar (*linear congruential generator* atau *LCG*). *LCG* adalah *PRNG* yang berbentuk:

$$X_n = (aX_{n-1} + b) \bmod m$$

X_n = bilangan acak ke-*n* dari deretnya

X_{n-1} = bilangan acak sebelumnya

a = faktor pengali

b = *increment*

m = modulus

Kunci pembangkit adalah X_0 yang disebut umpan (*seed*).

III. CIPHER BLOK JUMT

Cipher blok JUMT merupakan modifikasi dari blok cipher JAFT. Kebanyakan tekniknya mengadopsi dari teknik blok cipher JAFT. JUMT menggunakan kunci yang panjangnya 128-bit untuk proses enkripsi dan dekripsinya. Kunci yang diterima akan diubaha dalam bentuk heksadesimal dalam proses enkripsi. Kunci ini nantinya yang akan digunakan sebagai basis pembangkitan S-Box secara acak.

Blok plainteks pada cipher blok JUMT juga berukuran 128-bit dan juga dalam bentuk heksadesimal. Ukuran bloknya yakni 4x4 dan setiap sel terdapat satu heksadesimal yang merepresentasikan 8-bit atau satu karakter pada ASCII.

Secara garis besar cipher blok JAFT menggunakan prinsip-prinsip berikut:

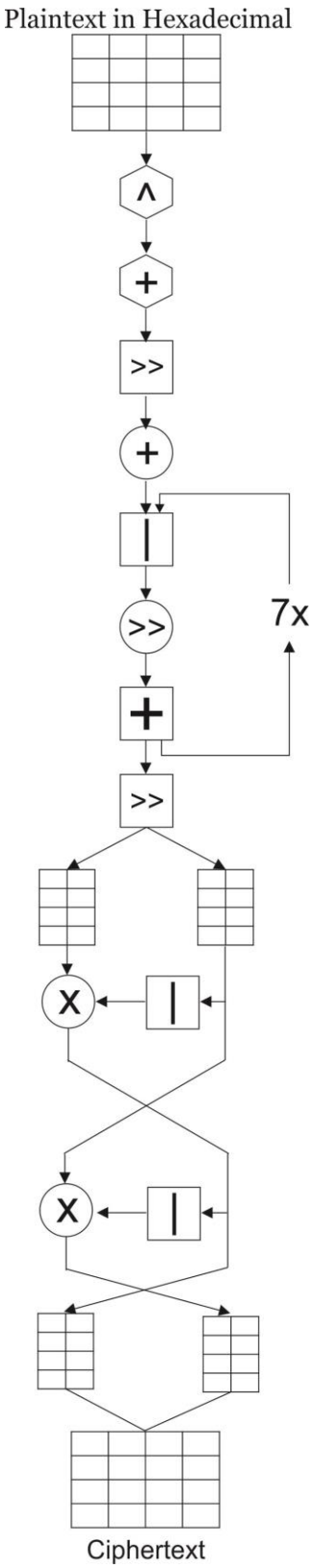
1. *randomSBox*. Pada awal tahapan enkripsi, S-Box yang digunakan dibangun secara random dari kunci yang diberikan.
2. *buildRoundKey*. Setelah terdapat S-Box yang dibangun secara acak saat pertama kali, maka *round key* akan dibangun secara otomatis dan langsung terdapat 7 *round key* yang digunakan pada proses *addRoundKey*.
3. *shiftCells*. Pada blok, masing masing sel digeser ke kanan sebanyak kunci tertentu yang diambil dari key 128-bit, yang diambil hanya satu sel heksadesimal di bagian pojok kiri atas.
4. *addKey*. Blok plainteks dilakukan operasi xor dengan kunci asli yang diberikan.
5. *addRoundKey*. Pada proses ini, kunci diambil dari kunci yang telah dibangun diawal, kemudian kunci tersebut dilakukan operasi xor dengan blok plainteks.
6. *subtituteSBox*. Pada blok plainteks isi dari masing-masing sel diganti dengan heksadesimal yang ada pada S-Box yang telah ditentukan terlebih dahulu.
7. *shiftRow*. Pada blok plainteks masing-masing baris dilakukan pergeseran. Pada baris pertama tidak dilakukan pergeseran, pada baris kedua digeser sebanyak 1 ke kanan, pada baris ketiga digeser sebanyak 2 ke kanan, dan pada baris keempat digeser sebanyak 3 ke kanan.
8. *xorOperation*. Merupakan operasi xor biasa.

A. Rancangan JUMT

Rancangan algoritma JUMT yakni sebagai berikut:

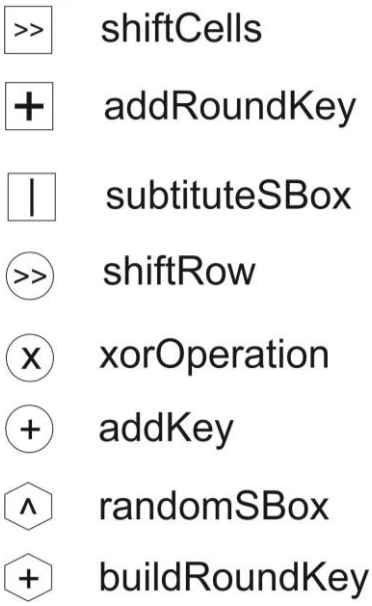
1. Pertama, plainteks diubah dalam bentuk heksadesimal dalam blok 4x4 yang berkuran 128-bit dan kunci juga diubah dalam bentuk heksadesimal.
2. Kunci kemudian digunakan untuk menghasilkan random S-Box pada proses *randomSBox*.
3. Setelah terdapat sebuah S-Box baru yang dibangun secara acak maka lanjut pada proses *buildRoundKey* untuk membangun *round key* sebanyak 7 buah yang digunakan pada proses putaran.
4. Pada blok plainteks kemudian diterapkan *shiftCells* dengan menggunakan sebuah key pada kunci.
5. Blok plainteks hasil dari proses 4 kemudian di xor dengan kunci asli yang terdapat di awal.
6. Setelah itu masuk pada proses *looping*, pertama blok plainteks hasil dari proses 5 di subtitusi dengan S-Box dengan proses *subtituteSBox*.
7. Selanjutnya hasil dari proses 6 diterapkan *shiftRow*.
8. Kemudian hasil dari proses 7 ditambahkan dengan *round key* yang sudah dibangun sebelumnya. *Round key* yang digunakan sesuai dengan hitungan pada putaran ke-i dengan i bilangan 1 sampai 7.
9. Setelah keluar dari *looping*, blok plainteks kemudian diterapkan kembali *shiftCells* namun dengan nilai key yang berbeda pada *shiftCells* diawal.
10. Kemudian blok plainteks dilewatkan pada jaringan Feistel. Pada jaringan Fesitel terdapat dua fungsi yakni subtitusi dengan S-Box dan operasi xor sederhana.
11. Setelah melewati jaringan Fesitel maka didapatkanlah hasil blok ciphernya. Blok blok ini kemudian digabung kembali (jika plainteks lebih dari 1 blok) untuk mendaptkan cipherteksnya. Inilah hasil cipherteks dari cipher blok JUMT.

Rancangan proses enkripsi cipher blok JUMT dapat dilihat pada gambar 3.



Gambar 3 : enkripsi JUMT

Keterangan:



Untuk proses dekripsi bisa dengan membalikkan prosesnya saja, namun dengan fungsi *randomSBox* dan *buildRoundKey* diletakkan diawal karena ini merupakan syarat awal untuk proses enkripsi maupun dekripsi dari blok cipher JUMT.

B. Pembangunan S-Box

Sbox dibangun dengan menggunakan prinsip pembangunan bilangan acak. Bilangan acak yang dihasilkan merupakan bilangan acak semu karena pembangunan bilangan acaknya tergantung dari kunci yang diberikan.

Berikut formula untuk pembangkitan bilangan acak:

$X_n = ((7X_{n-1} + 13) \bmod 256) + key_n$

Pada key karena jumlahnya tidak selalu 256 maka key akan diulang jika sudah mencapai panjang dari key tersebut maka akan kembali pada key_0.

Berikut merupakan gambaran dari prosedur untuk mengacak isi SBox.

```
public void randomSBox(string key){
  int[] X = new int[256];
  X[0] = key[0];
  SBox[0] = X[0];

  for (int n = 1; n < 256; n++){
    // assign random value
    X[n] = ((7*X[n-1] + 13) % 256) +
key[n%(key.Length)];

    // cek apakah sudah X[n] dalam SBox
    while (sudahAdaXndiSBox(X[n]){
      X[n]++;
      if (X[n]==(0xFF+1)){
            X[n] = 0x00;
```

```
        }
      }

      // letakkan nilai X[n] pada SBox
      SBox[n-1] = X[n];
    }
}
```

C. Pembangunan Round Key

Setelah mendapatkan SBox yang baru maka dilanjutkan dengan pembangunan *round key*. *Round key* langsung dibangun sebanyak 7 buah. Pada setiap putaran ke-i maka digunakan *round key* ke-i.

Pembangunan *round key* menggunakan algoritma sebagai berikut:

```
private void roundKey(int i){
 char temp1, temp2;

 for (int j = 0; j < blockKey[i].key.Length; j += 4)
{
   temp1   =   (char)(blockKey[i-1].key[j]^blockKey[i-
1].key[j+3]);
   for (int k = 0; k < (j + 3); k++)
   {
     temp2 = blockKey[i-1].key[k];
     blockKey[i].key[k] = blockKey[i-1].key[k+1];
     blockKey[i].key[k+1] = temp2;
     }
  }
}
```

D. Pengaturan shiftCells

Blok plainteks pada awalnya tersusun dengan rapi sesuai dengan urutannya, namun sebelum masuk pada proses enkripsi kita terlebih dahulu menggeser urutan pada blok plainteks. Proses *shiftCells* sama dengan proses pada cipher blok JAFT.

Untuk *shiftCells* yang pertama, nilai n didapat dengan formula

n = ((int) key1 mod 15) + 1

Untuk *shiftCells* yang kedua, nilai n didapat dengan formula

n = ((int) key7 mod 15) + 1

key1 berarti *key* pertama sebelum putaran dilakukan, diambil yang pojok kiri atas. *key7* berarti *key* pada putaran ke-7, untuk heksadesimal yang diambil merupakan heksadesimal di posisi pojok kanan atas.

Formula di atas menjamin akan adanya pergeseran pada blok plainteks, minimal terjadi satu pergeseran dan maksimal terjadi 15 pergeseran.

E. Add Key

Pada proses ini blok plainteks dilakukan operasi xor dengan kunci asli yang diberikan pertama kali. Kunci asli dalam bentuk heksadesimal yang sudah dibuat blok begitu juga dengan plainteks.

F. Penjadwalan Round Key

Kunci putaran sudah ada dan dibangun diawal dan kunci putaran itu lah yang digunakan dalama *looping*. *Round key* yang digunakan sesuai dengan putaran ke-i yang sedang dilakukan. Ini berarti *round key* ke-i digunakan untuk operasi pada putaran ke-i.

IV. EKSPERIMEN DAN PEMBAHASAN HASIL

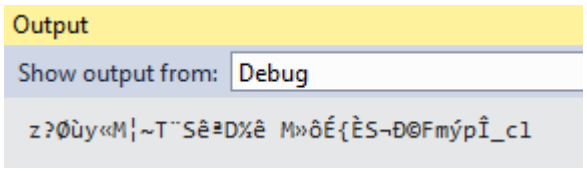
Untuk menguji algoritma yang telah dibuat dilakukan uji coba enkripsi pesan. Pengujian didapatkan hasil sebagai berikut:

Pengujian pertama:

pesan teks : “hari ini sangat cerah untuk beraktivitas”

kunci : “semangat”

hasil enkripsi :



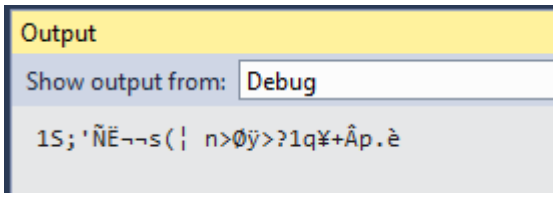
Gambar 4 : hasil enkripsi percobaan pertama

Pengujian kedua:

pesan teks : “inilah kehidupan, bersyukurlah”

kunci : “berjuanglah”

hasil enkripsi :



Gambar 5 : hasil enkripsi percobaan kedua

Setelah dilakukan percobaan, dihasilkan cipherteks seperti diatas. Tidak ada hubungan yang bisa dilihat antara plainteks, kunci maupun cipherteksnya. Kemunculan karakter menjadi sama sekali tidak terduga. dihasilkan cipherteks seperti diatas. Tidak ada hubungan yang bisa dilihat antara plainteks, kunci maupun cipherteksnya. Kemunculan karakter menjadi sama sekali tidak terduga. Berdasarkan pengujian enkripsi yang dilakukan, karakter yang muncul menjadi tidak beraturan bahkan untuk huruf yang sama memiliki cipherteks yang berbeda-beda. Ini tentunya akan meningkatkan keamanan dari pesan karena tidak terlihat suatu pola pada cipherteks.

V. ANALISIS KEAMANAN

Pada bagian ini penulis akan menajarkan bagaimana kemanan dan ketahanan blok cipher JAFT terhadap serangan dari kriptanalis. Aspek keamanan akan dianalisis dari berbagai sisi yakni analisis *key space*, analisis *known plaintext attack*, analisis keacakan, *linear cryptanalisis*, dan *differential cryptanalisis*.

A. Analisis key space

Key space merupakan nilai yang menyatakan banyaknya jumlah bit yang digunakan untuk mengeenkripsi suatu plainteks. Ukuran dari kunci haruslah cukup besar untuk bisa menghindari *brute-force attack*. Pada algoritma blok cipher JUMT ukuran kunci yang digunakan adalah 128-bit yang berarti ada sekitar 2^{128} atau sebanyak 3,4 x 10^{38} kemungkinanan pada kuncinya. Jika sebuah komputer dengan kemampuan prosesor mampu mengeksekusi satu juta kunci dalam satu detik maka akan dibutuhkan waktu untuk memecahkannya sekitar 3,4 x 10^{32} detik atau sekitar 1,08 x 10^{27} tahun yang dibutuhkan hanya untuk memecahkan kuncinya saja.

Dari analisis *key space*, algoritma blok cipher JUMT yang digunakan sangat aman karena dengan *brtue-force attack* butuh waktu yang sangat lama untuk memecahkannya.

B. Analisis known plaintext attack

Dalam serangan ini, kriptanalis menggunakan plainteks yang sudah dimiliki sebelumnya untuk mendapatkan kunci. Untuk kasus mode operasi ECB pada plainteks, analisis untuk mengetahui pola dari setiap blok akan samar-samar karena pada algoritma blok cipher JUMT setiap bit posisinya diubaha terus seiring dengan putaran yang dilakukan dan juga diterapkan subtitusi terhadap SBox sehingga tidak dapat dianalisa bagaimana pola dari kunci.

Karena posisi dari bit yang selalu berubah dan diganti setiap putaran maka menyebabkan algoritma blok cipher JUMT tidak dapat diserang secara statistik menggunakan analisis frekuensi. Frekuensi kemunculan tidak berpengaruh sama sekali terhadap cipherteks yang dihasilkan. Dapat disimpulkan algoritma blok cipher JUMT aman dari *known plaintext attack*.

C. Analisis keacakan

Saat sebelum enkripsi dilakukan, algoritma blok cipher JUMT terlebih dahulu membangun SBox sendiri berdasarkan kunci yang diberikan. SBox akan teracak sesuai dengan kunci yang diberikan, kemungkinan acak dari SBox ini adalah 16^{16} atau sekitar 1,84 x 10^{19} kemungkinan. Sedangkan dari kuncinya sendiri yang 128-bit butuh waktu sekitar 1,08 x 10^{27} tahun untuk dipecahkan ditambah dengan SBox yang juga diacak berdasarkan kunci. Ini menambah kerumitan dari blok cipher ini yang berarti secara kesuluruhan ada sekitar 6,256 x 10^{57} kemungkinan untuk memecahkan kunci dan SBox yang benar.

Dari sisi keacakan dan besarnya kemungkinan yang ada ini menyebabkan blok cipher sangat sulit dan bahkan hampir tidak mungkin untuk dipecahkan.

VI. KESIMPULAN DAN SARAN

Blok Cipher JUMT merupakan varian baru dari blok cipher yang sudah sebelumnya yakni Cipher Blok JAFT. Blok cipher JUMT merupakan modifikasi dan pengembangan dari blok cipher JAFT. Pada blok cipher JUMT menggunakan 128-bit kunci dan juga menggunakan 128-bit blok plainteks. Pada blok cipher JUMT SBox dibangun secara acak berdasarkan kunci yang diberikan dan dari SBox yang dibangunan ini kemudian dibangun *round key*.

Dari analisis keamanan, keamanan pada blok cipher JUMT jauh lebih baik dari blok cipher JAFT karena SBox yang selalu berubah dan pembangunan *round key* yang selalu berbeda berdasarkan SBox yang dibangun dari kunci yang diberikan.

Blok cipher JUMT masih bisa dikembangkan lebih jauh lagi karena ini masih dalam bentuk 128-bit. Kedepannya diharapkan bisa untuk memperbesar kunci dan plainteksnya menjadi 192-bit, 256-bit ataupun mencapai 512-bit nantinya, tergantung dari kebutuhan kemanannya seperti apa.

REFERENSI

[1] Mario Tressa Juzar, Rama Febriyan, dan Ahmad, “Cipher Blok JAFT”.

[2] Joan Daemen and Vincent Rijmen, “The Block Cipher Rijndael”.

[3] Paulo S.L.M. Barreto and Vincent Rijmen, “The KHAZAD Legacy-Level Block Cipher”.