ARES:
Alvaro-Rachman Encryption System

Devin Alvaro
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung
40132, Indonesia
13515062@std.stei.itb.ac.id

Edwin Rachman
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung
40132, Indonesia
13515042@std.stei.itb.ac.id

***Abstract*—Block cipher standar saat ini, Advanced Encryption Standard (AES) memiliki kekurangan dibanding Data Encryption Standard (DES), yaitu tidak menerapkan struktur Feistel. Konsekuensi hal ini adalah pada AES, dibutuhkan fungsi dekripsi yang berbeda dengan fungsi enkripsinya. Di sisi lain, DES menerapkan struktur Feistel, namun tidak lagi aman terutama karena panjang kuncinya yang pendek (56-bit). Algoritma block cipher kami, ARES, menerapkan struktur Feistel namun tetap aman, salah satunya karena panjang kuncinya 256-bit atau lebih. Ditambah lagi, ARES menerapkan operasi random transformations untuk meningkatkan kompleksitas serta confusion dan difusion dari enkripsi.**

Keywords—Feistel, Substitution, Random Transformations, Permutations, Block Cipher

I. PENDAHULUAN

Block cipher adalah algoritma untuk mengenkripsi dan mendekripsi informasi berupa kumpulan *bit* yang biasa disebut *block*. Pada dunia digital ini, *block cipher* memegang peranan penting dalam mengamankan informasi digital yang memang berupa kumpulan *bit*.

Selama beberapa dekade, Data Encryption Standard (DES) digunakan sebagai algoritma *block cipher* standar. Namun, seiring perkembangan teknologi terutama dengan meningkatnya kecepatan komputasi komputer, di masa sekarang DES dianggap sudah tidak lagi aman karena dapat dipecahkan dalam beberapa jam saja. Salah satu faktor utamanya adalah panjang kunci DES yang relatif pendek (56-bit).

Akhirnya DES digantikan oleh standar *block cipher* baru, yaitu Advanced Encryption Standard (AES). Hingga saat ini AES masih dianggap aman dengan panjang kunci 256-bit, akan tetapi tidak ada jaminan keamanan AES akan bertahan selamanya. Oleh karena itu, inovasi-inovasi di bidang kriptografi terutama pada *block cipher* penting untuk terus berjalan.

Salah satu kelebihan dari DES adalah struktur Feistel yang dimilikinya. Struktur Feistel adalah suatu skema *block cipher* sedemikian rupa sehingga fungsi untuk mengenkripsi informasi sama atau identik dengan fungsi dekripsinya. Perbedaannya hanyalah pada dekripsi, kunci untuk enkripsi dibalik (*reversed*). Di samping itu, meskipun lebih aman, AES tidak menerapkan struktur Feistel sehingga dibutuhkan algoritma dekripsi yang berbeda dengan algotima enkripsi.

Algoritma *block cipher* kami, ARES, menggabungkan kelebihan dari DES dan AES, yakni menerapkan struktur Feistel namun tetap aman, salah satunya karena menggunakan ukuran kunci 256-bit atau lebih. Secara singkat, algoritma ini menerapkan operasi-operasi AES dalam *round function* dari struktur Feistel. Untuk meningkatkan kompleksitas algoritma, ARES menggunakan operasi *random transformations* untuk meningkatkan *confusion* dan *difusion* agar algoritma lebih sulit dipecahkan. Penjelasan lebih lengkap tentang ARES ada pada Bab III.

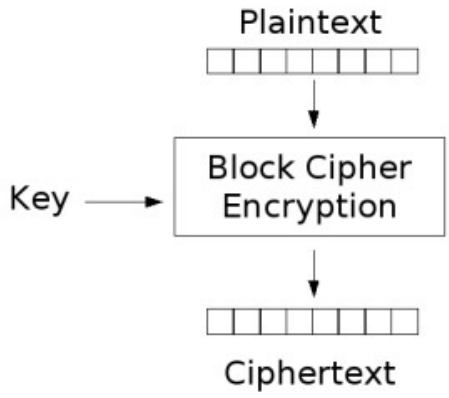
II. DASAR TEORI

A. Block cipher

Block cipher adalah algoritma enkripsi dan dekripsi kunci simetris deterministik yang beroperasi pada kumpulan *bits* dengan panjang tetap. Karena sifatnya yang beroperasi pada *bits*, *block cipher* memegang peranan penting dalam keamanan dan pengamanan informasi di dunia digital.

Pada dasarnya, *block cipher* terdiri dari sepasang algoritma, yaitu enkripsi dan dekripsi. Pada enkripsi, *block cipher* menerima input berupa *plaintext* bernilai biner (0 atau 1) dan sebuah key yang bernilai biner juga, menghasilkan *ciphertext* bernilai biner. Begitu juga pada dekripsi, namun posisi *ciphertext* dan *plaintext* dibalik. [2]

Berikut adalah gambaran enkripsi dengan *block cipher*:



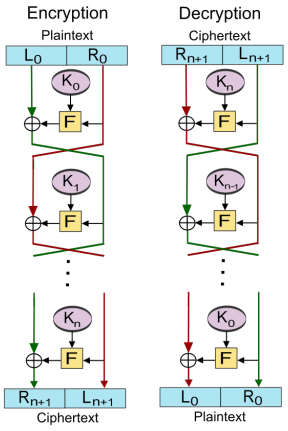
Gambar 1: Ilustrasi enkripsi *block cipher*
Sumber: https://upload.wikimedia.org/wikipedia/commons/0/0c/Encryption.png

B. *Struktur Feistel*

Struktur Feistel adalah suatu struktur simetris yang digunakan untuk mengkonstruksi *block cipher*. *Block cipher* yang menggunakan struktur Feistel memiliki properti simetris, yakni fungsi enkripsi dan dekripsinya identik atau bahkan sama. Biasanya, perbedaannya terletak pada *key*, di mana pada dekripsi, *key*-nya adalah *key* enkripsi yang dibalik. [2]

Cara kerja struktur Feistel terbagi-bagi menjadi beberapa *round*. Pada setiap *round*, *block* input dibagi menjadi 2. Kemudian, setengah *block* pertama dijadikan masukan ke *round function* F bersama dengan kunci K. Hasil dari *round function* tersebut kemudian di-XOR dengan setengah *block* kedua. Setelahnya, hasilnya dijadikan input *round function* berikutnya, dan begitu seterusnya.

Berikut adalah ilustrasi untuk memperjelas cara kerja Feistel:



Gambar 2: Ilustrasi struktur Feistel
Sumber: https://upload.wikimedia.org/wikipedia/commons/f/fa/Feistel_cipher_diagram_en.svg

C. *Confusion*

Confusion dalam *block cipher* berarti setiap *bit* dari *ciphertext* hasil harus tergantung dengan beberapa bagian dari kuncinya. Hal tersebut mengelabui hubungan antar *ciphertext* dan kunci. [3]

D. *Diffusion*

Diffusion dalam *block cipher* berarti jika satu *bit* pada *plaintext* diubah, maka secara statistik sekitar setengah dari semua *bit* dari *ciphertext* akan berubah. Juga berlaku sebaliknya, jika satu *bit* pada *plaintext* diubah, maka sekitar setengah dari semua *bit* dari *plaintext* akan berubah. [3]

E. *Substitution Box*

Substitution box (S-box) adalah metode yang digunakan untuk melakukan substitusi yang memetakan angka n-bit ke angka m-bit. Untuk *block cipher*, S-box digunakan untuk menimbulkan properti Shannon confusion. [4]

F. *Permutation Box*

Permutation Box (P-box) adalah metode *bit-shuffling* yang dilakukan untuk melakukan permutasi atau transposisi *bit* input sehingga outputnya tersebar antar S-box. Hal tersebut menimbulkan properti Shannon diffusion. [4]

III. Rancangan Algoritma

A. *Struktur Feistel*

Algoritma yang dikembangkan menggunakan struktur Feistel dalam proses enkripsi dan dekripsinya. Struktur Feistel menerima input berupa plaintext block 256-bit untuk ronde i (M_i) yang dipisahkan menjadi dua half-block 128-bit kiri (L_i) dan kanan (R_i). Half-block kanan kemudian diaplikasikan round function (F) dengan parameter round key 128-bit untuk ronde saat itu (K_i). Hasil dari round function tersebut kemudian di-XOR-kan dengan half-block kiri. Setelah itu, half-block kiri dan kanan ditukar dan langkah sebelumnya diulang lagi hingga 16 ronde. Di akhir ronde terakhir half-block kiri dan kanan digabung kembali sehingga menghasilkan output adalah ciphertext 256-bit. Untuk dekripsi hal yang serupa dilakukan, tetapi dengan urutan terbalik.

B. *Key Schedule*

Key schedule yang digunakan untuk pembangkitan round key (Ki) menggunakan fungsi yang sama seperti yang digunakan untuk round key (F(Mi, Ki) = Ci+1), tetapi Mi diganti dengan 128-bit least significant dari key (Key0), dan Ki diganti dengan 128-bit selanjutnya dari key (Key1), sehingga K0 = F(Key0, Key1). Jika key yang digunakan 374-bit, maka K1 = F(Key1, Key2). Begitu pula untuk key berukuran (256+n*128)-bit selanjutnya. Setelah bit pada key habis, maka digunakan round key yang dihasilkan dari ronde ke-0. Misal

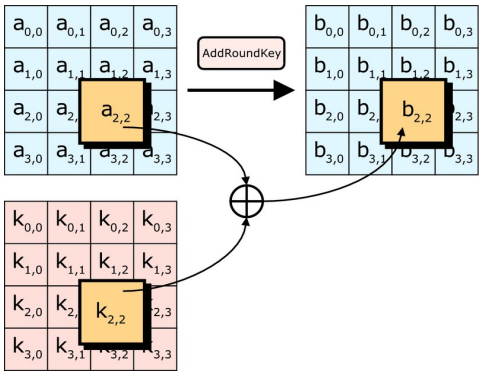
untuk key 374-bit sebelumnya: K2 = F(Key2, K0); K3 = F(K0, K1); K4 = F(K1, K2); dan seterusnya hingga semua round key terbangkitkan.

C. *Round Function*

Langkah-langkah yang diaplikasikan pada input half-block 128-bit pada round function yang digunakan oleh struktur feistel algoritma ini adalah sebagai berikut:

1. *Key Mixing*

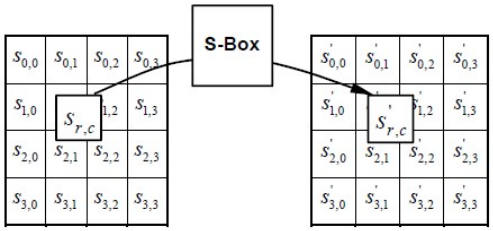
Plainteks half-block 128-bit di-XOR-kan dengan round key 128-bit ($X_i = M_i \oplus K_i$). Langkah ini seperti langkah AddRoundKey pada algoritma Rijndael, seperti yang diilustrasikan pada ilustrasi di bawah:



Gambar 3: Ilustrasi langkah Key Mixing
Sumber: https://upload.wikimedia.org/wikipedia/commons/0/03/AES-AddRoundKey.png

2. *Substitution*

Hasil 128-bit dari key mixing disubstitusikan menggunakan 16 S-box random yang dihasilkan menggunakan seed key awal. Satu S-box diperuntukkan untuk satu byte dari half-block pesan. Langkah ini diilustrasikan seperti pada ilustrasi di bawah:



Gambar 3: Ilustrasi langkah Substitution
Sumber: http://imps.mcmaster.ca/courses/SE-4C03-07/wiki/siaa/se4c03_aes_wiki(7)_html_4c9f4601.gif

3. *Random Transformations*

Setiap byte ke-j round key ($K_i[j]$, dimana j=0 adalah byte least significant dan j=15 adalah byte most significant) akan digunakan untuk menentukan salah satu dari 8 jenis transformasi random ($T(Y_i, K_i[j])$) yang akan diaplikasikan ke hasil substitution. Hasil dari transformasi selanjutnya kemudian diaplikasikan ke transformasi selanjutnya dan seterusnya, sehingga hasilnya:

$Z_i = T(T(T(T(T(T(T(T(T(T(T(T(T(T(Yi, K_i[0]), K_i[1]), K_i[2]), K_i[3]), K_i[4]), K_i[5]), K_i[6]), K_i[7]), K_i[8]), K_i[9]), K_i[10]), K_i[11]), K_i[12]), K_i[13]), K_i[14]), K_i[15])$.

- Jika $K_i[j]$ berada dalam range 0x00-0x0F dan 0xF0-0xFF, maka tidak ada transformasi yang dilakukan.

- Jika $K_i[j]$ berada dalam range 0x10-0x2F, maka akan dilakukan transformasi ShiftRows seperti pada Rijndael.

00	01	02	03		00	01	02	03
04	05	06	07	→	07	04	05	06
08	09	0A	0B		0A	0B	08	09
0C	0D	0E	0F		0D	0E	0F	0C

- Jika $K_i[j]$ berada dalam range 0x30-0x4F, maka akan dilakukan transformasi ShiftRows seperti pada Rijndael tetapi dengan urutan terbalik.

00	01	02	03		00	01	02	03
04	05	06	07	→	05	06	07	04
08	09	0A	0B		0A	0B	08	09
0C	0D	0E	0F		0F	0C	0D	0E

- Jika $K_i[j]$ berada dalam range 0x50-0x6F, maka akan dilakukan rotasi ke kiri setiap byte pada hasil sebesar 3-bit terakhir dari $K_i[j]$.

  Misal $K_i[j]$ = 0x52 maka rotasi kiri 2 kali, byte 0b10011011 menjadi 0b01101110.

- Jika $K_i[j]$ berada dalam range 0x70-0x8F, maka akan dilakukan rotasi ke kanan setiap byte pada hasil sebesar 3-bit terakhir dari $K_i[j]$.

  Misal $K_i[j]$ = 0x82 maka rotasi kanan 2 kali, byte 0b10011011 menjadi 0b11100110.

- Jika $K_i[j]$ berada dalam range 0x90-0xAF, maka posisi setiap byte pada kotak 4x4 byte kemudian digeserkan berputar searah jarum jam sebesar 4-bit terakhir dari $K_i[j]$.

  Misal $K_i[j]$ = 0x91, maka rotasi byte akan seperti berikut (kiri atas byte least significant, kanan bawah byte most significant):

00	01	02	03		04	00	01	02
04	05	06	07	→	08	09	05	03
08	09	0A	0B		0C	0A	06	07
0C	0D	0E	0F		0D	0E	0F	0B

- Jika $K_i[j]$ berada dalam range 0xB0-0xCF, maka posisi setiap byte pada kotak 4x4 byte kemudian digeserkan berputar berlawanan jarum jam sebesar 4-bit terakhir dari $K_i[j]$.

  Misal $K_i[j]$ = 0xC1, maka rotasi byte akan seperti berikut (kiri atas byte least significant, kanan bawah byte most significant):

00	01	02	03		01	02	03	07
04	05	06	07	→	00	06	0A	0B
08	09	0A	0B		04	05	09	0F
0C	0D	0E	0F		08	0C	0D	0E

- Jika $K_i[j]$ berada dalam range 0xD0-0xEF, maka akan dilakukan penjumlahan modulo hasil dengan hasil dari n transformasi sebelumnya, dimana n adalah 4-bit terakhir dari $K_i[j]$. Jika n lebih besar dari jumlah transformasi yang sudah pernah dilakukan atau transformasi ini merupakan transformasi pertama, maka tidak ada transformasi yang dilakukan.

4. Permutation

Setiap bit kemudian dipermutasikan menggunakan sebuah P-box berukuran 128-bit yang dihasilkan secara acak berdasarkan kunci awal (C_{i+1} = $P(Z_i)$). P-box tersebut dihasilkan sedemikian rupa sehingga setiap 16 bit dari 2 byte akan tersebar ke setiap 16 S-box pada iterasi selanjutnya.

D. Mode operasi

Block cipher ini melakukan enkripsi dan dekripsi dalam satuan blok 256-bit. *Block cipher* dapat dioperasikan dengan mode operasi adalah mode-mode berikut:

1. Electronic Code Block (ECB)

Setiap blok dienkripsi dan didekripsi secara independen. Mode ini kurang baik untuk dipakai karena tidak ada efek difusi pada plainteksnya.

2. Cipher Block Chaining (CBC)

Setiap blok plainteks di-XOR dengan hasil blok cipherteks sebelum dilakukan enkripsi untuk blok tersebut. Untuk blok pertama digunakan sebuah initialization vector yang disediakan pengguna.

3. Cipher Feedback (CFB)

Setiap blok hasil cipherteks di-XOR dengan blok plainteks sebelum dilakukan enkripsi untuk blok tersebut. Untuk blok pertama digunakan sebuah initialization vector yang disediakan pengguna.

4. Output Feedback (OFB)

Seperti CFB tetapi yang digunakan sebagai feedback block adalah hasil sebelum di-XOR.

IV. Implementasi dan Eksperimen

A. Implementasi

Berikut adalah cuplikan *pseudocode encrypt* dan *decrypt* dari implementasi algoritma ARES.

```
HALF_BLOCK_LENGTH = 16
ROUND_COUNT = 16

def encrypt(block):
    left = block[:HALF_BLOCK_LENGTH]
    right = block[HALF_BLOCK_LENGTH:]

    for i in range(ROUND_COUNT):
        left = right
        right = (c ^ d) for c, d in zip(
                left,
round_function(right, round_keys[i]))

    return left + right

def decrypt(block):
    left =block[:HALF_BLOCK_LENGTH]
```

```
    right =  block[HALF_BLOCK_LENGTH:]

    for i in
reversed(range(ROUND_COUNT)):
        right = left
        left = (c ^ d) for c, d in zip(
                right,
self.round_function(left,
round_keys[i])))

    return left + right

def round_function(half_block,
round_key):
    mixed = mix_keys(half_block,
round_key)
    subtituted = substitute(mixed)
    transformed = transform(substituted)
    permuted = permute(transformed)
    return permute
```

Seperti dapat dilihat, fungsi *encrypt* menerima masukan berupa block, kemudian membaginya menjadi *left* dan *right*. Kemudian, untuk setiap *round*, dilakukan *round function* terhadap *left* dan *right*. Cara kerja yang sama juga berlaku pada *decrypt* namun bekerja kebalikannya.

Seperti yang dijelaskan pada rancangan algoritma, *round function* ARES melakukan *mix keys*, substitusi, transformasi, dan permutasi pada *half block*. Implementasi dari keempat fungsi tersebut tidak dimasukkan karena cukup panjang dan sudah dijelaskan di rancangan algoritma.

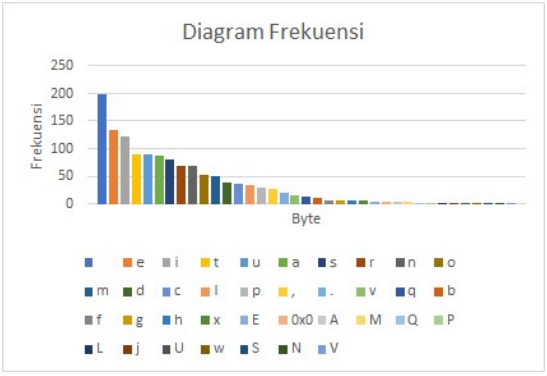
Source code lengkap dari implementasi algoritma ARES dapat dilihat pada repositori GitHub kami https://github.com/devinalvaro/kripto-ares.

B. Eksperimen

Plaintext yang kami gunakan untuk pengujian adalah sebagai berikut:

```
Lorem ipsum dolor sit amet, eu qui scaevola
dissentiet, mea at atqui referrentur, justo inani
cu has. […]
```

Frekuensi byte dari *plaintext* dapat dilihat pada diagram frekuensi di bawah:



Gambar 4: Diagram Frekuensi *Plaintext*

Dapat diperhatikan bahwa *range* dari frekuensi byte berkisar dari 198 hingga 1. Karakter spasi paling sering muncul dengan frekuensi 198, diikuti dengan karakter ‘e’ dengan frekuensi 134, karakter ‘i’ dengan frekuensi 122, dan seterusnya. Jumlah byte unik yang ada adalah 38.

Algoritma enkripsi kami kemudian kami uji dengan beberapa kasus.

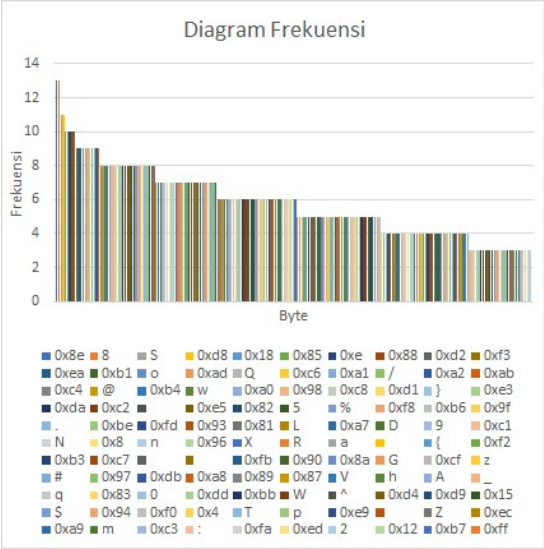
Kasus 1

Kondisi awal

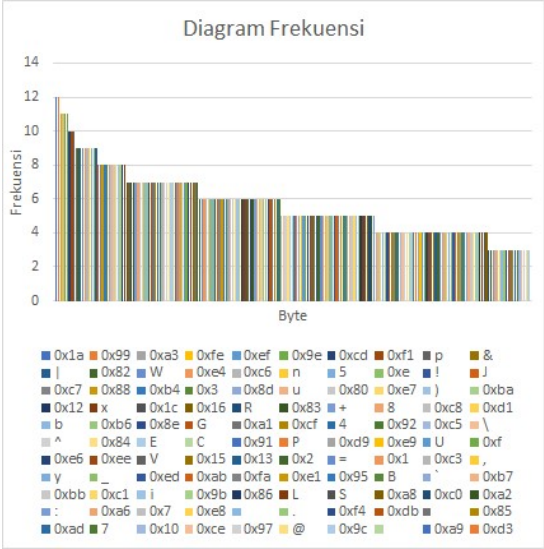
Key: *argarg3r9i2p9tuslernsbgaeiga4tq2*

Ciphertext:
```
\xc4\xa8\x84\x14@t\x18\xb4\x89\xff\x87wL
\xacKV\xf3\x8c\x8ehA8\xa0\tuSwKy_\xa7\x98VD
\xb5\xea\xb1K\x189o\x18\xc1\xd3\xad\xc8
\xc8q\xd1} […]
```

Frekuensi byte dari *ciphertext* kasus 1 dapat dilihat pada diagram frekuensi di bawah:



Gambar 5: Diagram Frekuensi *Ciphertext* Kasus 1



Gambar 6: Diagram Frekuensi *Ciphertext* Kasus 2

Dapat diperhatikan bahwa *range* dari frekuensi byte berkisar dari 14 hingga 1, jauh lebih kecil dibanding plainteks. Jumlah byte unik juga lebih banyak dibanding plaintext, yaitu hingga 256 byte.

Kasus 2

Huruf pertama plaintext diubah

Key: *argarg3r9i2p9tuslernsbgaeiga4tq2*

Ciphertext:
`\x10\x1a\x1c\xb3\x16RI\xff\x83+\xce\x02=8`
`\x01\xcd\xc2\x97@\x9c\xc8\x9a\xcd\xfd`
`\xc3w&\xd1&,\xc2\xa9y\xd3\x89J|\xd3_\x81`
`\xd8\xf5k\x01 […]`

Jumlah bit yang berubah dari ciphertext kasus 1 dan kasus 2 adalah **5270/10496 (50.20960365853659 %)**.

Frekuensi byte dari *ciphertext* kasus 3 dapat dilihat pada diagram frekuensi di bawah:

Kasus 3

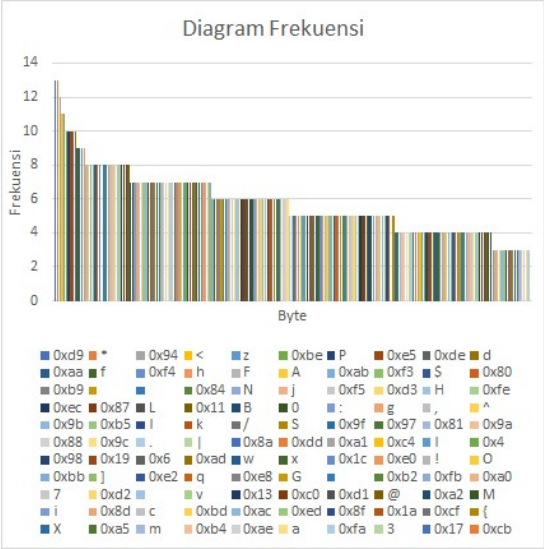
Karakter pertama key diganti

Key:
brgarg3r9i2p9tuslernsbgaeiga4tq2

Ciphertext:
`l\x8c\xd3\x08Z\xeb\xbb\xdc\x9a\x03\xfeG`
`\xe8\xae\xc9\xf3\x14`\x04\xb5O\x01$\x03ng`
`\xdd]\xb3\x87\xbe\x92` **[…]**

Jumlah bit yang berubah dari ciphertext kasus 1 dan kasus 3 adalah **5167/10496 (49.228277439024396%)**.

Frekuensi byte dari *ciphertext* kasus 3 dapat dilihat pada diagram frekuensi di bawah:



Gambar 7: Diagram Frekuensi *Ciphertext* Kasus 3

V. ANALISIS

A. Analisis brute force attack

Algoritma ARES menggunakan kunci dengan panjang minimal 256-*bit*, tepatnya 256+128*n bit*.

Untuk analisis ini, anggaplah digunakan panjang kunci minimal, yaitu 256-*bit*, berarti terdapat $2^{256} = 1.16 \times 10^{77}$ kemungkinan kunci. Kemungkinan sebanyak itu apabila di-*brute force* oleh superkomputer tercepat saat ini saja, *Sunway TaihuLight* milik Tiongkok yang berkecepatan 93 petaflops (93×10^{15} operasi per detik), masih membutuhkan waktu sangat lama untuk memecahkan ARES secara brute force.

Berikut perhitungannya:

t = jumlah operasi / kecepatan komputasi
$t = 2^{256} / 93 \times 10^{15}$
$t = 1.16 \times 10^{77} / 93 \times 10^{15}$
$t = 1.24 \times 10^{60}$ detik

Waktu tersebut bila diubah ke satuan tahun adalah 3.92×10^{52} tahun, lebih dari perkiraan umur alam semesta.

Ditambah lagi, seperti yang disebutkan sebelumnya ARES juga dapat menggunakan kunci lebih dari 256-*bit*, tepatnya 256+128*n bit*. Oleh karena itu, ARES dapat dianggap aman terhadap *brute force attack*.

B. Frequency analysis attack

Jika diagram frekuensi *plaintext* pada Gambar 4 dan diagram frekuensi *ciphertext* pada Gambar 5 atau 6 dibandingkan, korelasi antara frekuensi keduanya hampir tidak ada.

Jumlah byte unik pada *plaintext* hanya 38 dengan *range* frekuensi hingga 198. Diagram frekuensinya juga terlihat tidak rata.

dan jumlah byte unik pada *ciphertext* sampai 256 (seluruh kemungkinan byte) dengan *range* frekuensi hanya hingga 15, jauh lebih kecil dan rata dibandingkan pada *plaintext*.

Oleh karena itu, dapat disimpulkan bahwa seorang penyerang akan sangat kesulitan untuk melakukan *frequency analysis attack* terhadap ARES sehingga ARES dapat dianggap aman terhadap *frequency analysis attack.*

C. Confusion

Pada eksperimen, kami menunjukkan bahwa dengan mengganti sedikit bagian plaintext dari kasus 1 ke kasus 3, terjadi perubahan sebesar 49.22% pada ciphertext. Dari fakta tersebut, dapat disimpulkan algoritma ARES memiliki confusion yang tinggi.

D. Diffusion

Pada eksperimen, kami menunjukkan bahwa dengan mengganti sedikit bagian plaintext dari kasus 1 ke kasus 2, terjadi perubahan sebesar 50.20% pada ciphertext. Dari fakta tersebut, dapat disimpulkan algoritma ARES memiliki diffusion yang tinggi.

VI. KESIMPULAN

Algoritma block cipher ARES dapat menerapkan struktur Feistel sehingga fungsi enkripsi sama dengan dekripsi, namun tetap aman. Keamanan tersebut dikarenakan, di antaranya panjang kunci yang minimal 256-bit, tepatnya 256+128n bit sehingga ARES aman dari brute force attack.

ARES juga aman dari frequency attack karena berdasarkan eksperimen kami, diagram frekuensi ciphertext cukup rata dibandingkan dengan diagram frekuensi plaintext-nya yang tidak seimbang. Dari kedua diagram tersebut dapat disimpulkan tidak dapat dilakukan frequency pada ARES.

ARES memiliki confusion dan diffusion yang tinggi, disimpulkan dari hasil eksperimen yang menunjukkan bahwa sulit ditemukan hubungan antara plaintext, ciphertext, dan key. Hal ini dikarenan pada ARES, dengan mengubah sedikit plaintext atau key, dihasilkan ciphertext yang sangat berbeda.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. sebagai dosen mata kuliah IF4020 Kriptografi. Penulis juga berterima kasih kepada seluruh penulis yang tulisannya dijadikan referensi untuk makalah ini. Terakhir, penulis berterima kasih kepada orang tua, teman-teman, dan asisten mata kuliah ini atas dukungan dan bantuannya dalam menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi, 2005. Diktat Kuliah Kriptografi. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[2] Menezes, Alfred J, 2001, Handbook of Applied Cryptography (Fifth ed.), Waterloo: CRC Press. hal. 251

[3] Stallings, William, 2014, Cryptography and Network Security (6th ed.), Upper Saddle River, N.J.: Prentic Hall. hal. 67–68.

[4] Chalmers University of Technology, 2007, "Cryptography 2007", Gothenburg: Chalmers University of Technology Computer Science and Engineering (http://www.cs.chalmers.se/Cs/Grundutb/Kurser/krypto/lect03-2x2.pdf PDF diakses pada 16 Maret 2018).

[5] https://github.com/devinalvaro/kripto-ares diakses pada 15 Maret 2018.

PERNYATAAN

Dengan ini kami menyatakan bahwa makalah yang kami tulis ini adalah tulisan kami sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 April 2018

Devin Alvaro (13515062) Edwin Rachman (13515042)