HARS: 256-Bit Block Cipher Berbasis Jaringan Feistel Menggunakan Fungsi Putaran Rijndael dan Serpent

Muhammad Furqan Habibi (13511002) [1] dan Alifa Nurani Putri (13511074) [2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung g, Jl. Ganesha 10 Bandung 40132, Indonesia
[1] furqan.habibi1@gmail.com, [2] alifanuraniputri@gmail.com

***Abstrak*—Pada paper ini diajukan sebuah algoritma block cipher baru yang memadukan dua algoritma yanng memenangkan dua peringkat terbaik pada kompetisi AES (*Advance Encryption Standard*). Algoritma baru ini dinamakan HARS, yang dapat mengekripsi data dengan panjang blok dan kunci sejumlah 256-bit. HARS memiliki kelebihan dari segi kompleksitas namun tetap memiliki struktur yang sederhana sehingga mudah diimplementasikan dalam level *hardware*, HARS cocok digunakan untuk enkripsi data yang membutuhkan tingkat keamanan yang tinggi.**

Kata kunci—Kriptografi; Block-cipher; Rijndael; Serpent; Jaringan Feistel;

I. PENDAHULUAN

Perkembangan teknologi selalu beriringan dengan bertambahnya jumlah pengguna dan peredaran data. Meningkatnya kedua hal tersebut turut memicu tingginya resiko datangnya serangan (*attack*) yang mencgancam kerentanan suatu sistem. Berbagai teknik keamanan informasi dapat digunakan untuk mengurangi kesuksesan ancaman (*threat*) dan juga untuk melindungi aset yang dimiliki suatu sistem. Salah satu teknik yang banyak digunakan adalah kriptografi atau 'pesan tersembunyi'.

Aspek keamanan utama yang berusaha dijamin dengan teknik kriptografi adalah *confidentiality*, yaitu bagaimana menjaga kerahasiaan dari suatu aset. Aspek ini berusaha menjaga suatu aset agar tidak dapat diakses oleh pihak yang tidak diizinkan. Penggunaan kriptografi untuk menjaga kerahasiaan sudah dilakukan sebelum teknologi komputasi digunakan. Pada awalnya, kriptografi klasik menjaga kerahasiaan pesan dengan operasi karakter. Saat teknologi komputasi sudah maju, berkembangkah kriptografi modern yang digunakan hingga saat ini.

Kriptografi modern menjaga kerahasiaan pesan dengan proses komputasi, yaitu dengan metoda operasi bit. *Block-cipher* merupakan salah satu cabang dari kriptografi modern yang beroperasi pada sejumlah bit tertentu dalam satu block. Hingga saat ini penelitian untuk mengembangkan algoritma enkripsi dan dekripsi block-cipher masih sangat terbuka. Aspek kompleksitas dan kecepatan menjadi dua poin yang menjadi fokus utama pengembangan algoritma block-cipher baru. Masalah yang ingin diselesaikan adalah bagaimana kerahasiaan pesan dapat terjaga semaksimal mungkin, yaitu dengan mendesain suatu cara agar usaha untuk menyerang *confidentiality* suatu aset menjadi sangat besar dan melebihi nilai dari aset tersebut.

Pada tahun 2001 *National Institute of Standards and Technology* (NIST) mengeluarkan sebuah standar algoritma kriptografi baru yang dinamakan AES (*Advance Encryption Standard*) yang menggantikan sebuah standar lama yang sudah dinyatakan tidak aman yaitu DES (*Data Encryption Standard*) . Standar algoritm abaru ini ditentukan dari sebuah kompetisi terbuka.

Dalam kompetisi AES, Vincent Rijmen dan Joan Daemen mengajuakan sebuah algoritma bernama *Rijndael*, merujuk pada [2] yang mendukung engkripsi dengan panjang kunci 128-256 bit. Algoritma ini memenangkan kontes AES dan dinobatkan standar enkripsi baru yang digunakan hingga saat ini. Pada kontes yang sama, Ross Anderson, Eli Biham, dan Lars Knudsen juga mengebangkan sebuah algoritma bernama *Serpent*, merujuk pada [1] dan menempati pringkat kedua. Perbedaan utama Serpent dan Rijndael adalah jumlah *rounds* dan level enkripsinya, dimana algoritma Rijndael memiliki *rounds* yang lebih sedikit dan level enkripsi yang lebih rendah sehingga waktu eknripsi dan dekripsinya lebih cepat.

Kompleksitas dan kecepatan merupakan dua hal utama yang menentukan performansi suatu algoritma kriptografi. Pada awalnya, ketika kinerja *hardware* masih sangat terbatas dan proses komputasi membutuhkan waktu yang relatif lama, algoritma yang kompleks namun membutuhkan proses komputasi tinggi cenderung tidak dipilih. Kini, seiring dengan perkembangan pesat teknologi yang menjadikan proses komputasi dapat berlangsung jauh lebih cepat, algoritma yang kompleks menjadi lebih mudah diterima. Maka dari itu, berbagai macam algoritma kriptografi diciptakan dengan kompleksitas tinggi, untuk meningkatkan keamanan.

II. DASAR TEORI

A. *Algoritma Kriptografi Rijndael*

Rijndael merupakan sebuah algoritma kriptografi block-cipher dengan panjang block dan panjang kunci dapat berjumlah 128, 192, atau 256 bit. Masukan plaintext akan dipetakan sebagai array *state* dan masukan kunci akan dipetakan sebagai array *key*. Kedua komponen tersebut digunakan dalam setiap iterasi, hingga pada akhir iterasi ciphertext dihasilkan dari *state* akhir.

Pada setiap iterasi, yang disebut dengan *round*, terdapat suatu fungsi yang dinamakan *round function* dengan masukan berupa *state* dan *roundkey*. Round function inilah yang melakukan sejumlah transformasi untuk menghasilkan ciphertex. Round function merupakan suatu identitas yang membedakan suatu algoritma block-cipher iterasi dengan algoritma sejenisnya.

Algoritma Rijndael memiliki round function yang terdiri dari empat transformasi berbeda yaitu: *ByteSub*, *ShiftRow*, *MixColumn*, dan *AddRoundKey*. Keempat transformasi tersebut dilakukan pada setiap round hingga tepat sebelum round terakhir. Pada round terakhir mix column tidak dilakukan sehingga hanya terdapat tiga transformasi.

Berikut adalah penjelasan mengenai keempat transformasi tersebut:

1. ByteSub

   Transformasi ini merupakan operas substitusi byte non-linear. Substitusi dilakukan dengan menggunakan tabel bernama S-box. S-box digenerasi dengan dua buah transformasi yang secara detail dijelaskan pada [2].

2. ShiftRow

   Pada transformasi ini dilakukan pergeseran bytes secara siklik untuk setiap baris, dengan jumlah pergeseran yang berbeda. Baris ke-0 tidak diubah. Baris pertama digeser sejumlah C1 bytes, baris kedua sejumalh C2 bytes, dan baris ketiga digeser C3 bytes.

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

3. MixColumn

   Kolom-kolom dalam satu state dibentuk ke dalam sebuah polinomial dan dikalikan dengan modulo x^4+1 dengan sebuah polinomial $c(x)$

   $$c(x) = \text{`03'}\, x^3 + \text{`01'}\, x^2 + \text{`01'}\, x + \text{`02}$$

   Perkalian tersebut dapat dikategorikan sebagai perkalian matrix.

4. AddRoundKey

   Transformasi ini melibatkan kunci, yaitu kunci diaplikasikan pada state dengan operasi bitwise XOR .

Selain empat transformasi tersebut, dibutuhkan *Key Schedule* (penjadwalan kunci untuk menghasilkan RoundKey. Terdaoat dua komponen utama yaiyu key ecpansion dan rond key selection. Tiga prinsip yang digunakan dalam penjadwalan kunci ini adalah:

1. Jumalh bit RoundKey sama dengan panjang block dikalikan dengan nomor round ditambah 1.
2. Cipher Keyy di-*expand* menjadi ExpandedKey.
3. RoundKey diambil dari ExpandedKey, dengan mengambil sejumlah Nb *words* untuk setiap round berurutan dari awal *words* ExpandedKey.

B. *Algoritma Kriptografi Serpent*

Jika dibandingkan dengan Rijndael, Serpent memiliki waktu yang lebih lambat namun memberikan tingkat keamanan yang jauh lebih tinggi.

Serpent melakukan enkripsi terhadap 32-bit words (128-bit) plaintext P menjadi 128-bit ciphertext C di bawah kendali 33 buah subkey 128-bit K0,...,K32. Panjang dari key pengguna dapat bervariasi namun untuk pengajuan kontes AES ditetapkan sepanjang 128, 192, atau 256 bit. Key yang lebih pendek dari 256-bit akan dipetakan menjadi 256-bit dengan menambahkan bit '1' pada ujung MSB diikuti dengan bit '0' secukupnya hingga mencapai 256-bit.

P akan digunakan sebagai intermediate data pertama B0 yang merupakan input terhadap round pertama. Terdapat 32 round, dari round 0 hingga round 31. Output dari round 0 adalah B1, output dari round 1 adalah B2, output dari round i adalah Bi+1, demikian seterusnya hingga output dari round terakhir adalah B32. B32 kemudian dioutputkan menjadi ciphertext C. Setiap fungsi round Ri ($i \in \{0,...,31\}$) menggunakan hanya sebuah S-box yang direplikasi.

Jika dirangkum, maka setiap round pada serpent akan melalui langkah-langkah ini:

1. Key Mixing

   Pada setiap round, sebuah subkey 128-bit Ki di-XOR-kan dengan intermediate data sekarang Bi.

2. S-Boxes

   Kombinasi input dan Key sepanjang 128-bit dibagi-bagi menjadi kumpulan 4-bit. Setiap 4-bit kemudian diterapkan S-box yang sama menghasilkan 4-bit output. Pada akhirnya dihasilkan intermediate vector Si(Bi $\oplus$ Ki).

3. Transformasi Linear

   128-bit intermediate vector dipandang sebagai empat buah 32-bit word. Masing masing word kemudian dicampur secara linear dengan langkah-langkah berikut :

   X0, X1, X2, X3 := Si(Bi $\oplus$ Ki)

   X0 := X0 <<< 13

X2 := X2 <<< 3

X1 := X1 ⊕ X0 ⊕ X2

X3 := X3 ⊕ X2 ⊕ (X0 << 3)

X1 := X1 <<< 1

X3 := X3 <<< 7

X0 := X0 ⊕ X1 ⊕ X3

X2 := X2 ⊕ X3 ⊕ (X1 << 7)

X0 := X0 <<< 5

X2 := X2 <<< 22

Bi+1 := X0, X1, X2, X3

di mana <<< melambangkan rotasi, dan << melambangkan shift. Pada round terakhir, transformasi linear digantikan oleh proses ¬key mixing tambahan: B32 := S7(B31 ⊕ K¬31) ⊕ K32.

C. *Jaringan Feistel*

Feistel network atau Jaringan Feistel merupakan suatu desain konstruksi yang banyak digunakan oleh algoritma ekripsi simetris. Jaringan Feistel didefinisikan sebuagai sebuah fungsi $F: \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$. Terdapat parameter $d \in N$, dimana Round Function nya yaitu $f_1, ..., f_d : \{0,1\}^n \rightarrow \{0,1\}^n$. Terdapat d rounds, yang biasanya sejumlah 12-16. Pada setiap round i dilakukan sejumlah operasi berikut:

1. Input setiap round dibagi menjadi dua, yaitu L_{i-1} (kiri) dan R_{i-1} (kanan).
2. Round Functions diaplikkasikan pada bagian kanan, $f_i (R_{i-1})$.
3. Dilakukan operasi XOR dari nilai output fungsi dengan bagian kiri, $L_{i-1} \oplus f_i (R_{i-1})$, dan
4. Dilakukan pertukaran bagian kiri dan kanan yang menjadi output dari suatu round, $L_i \; || \; Ri := R_{i-1} \; || \; L_{i-1} \oplus f_i (R_{i-1})$.

III. RANCANGAN BLOCK-CIPHER

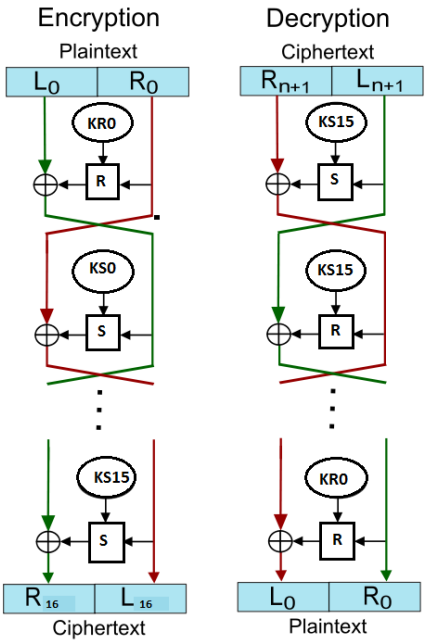
Algoritma ini pada dasarnya menggabungkan dua algoritma finalis lomba AES (Advanced Encryotion Standard) yaitu: Rijndael dan Serpent dengan menggunakan Jaringan Feistel (Feistel Network). Dengan menggabungkan kedua algoritma tersebut, tingkat kompleksitas dari algoritma ini lebih besar.

Seluruh proses dalam algoritma ini mengikuti struktur simetris dari jaringan Feistel. Dengan jaringan Feistel, operasi enkripsi dan Dekripsi cukup serupa. Sehingga dekripsi dapat dilakukan dengan melakukan reverse dari operasi eknripsinya.

Algoritma ini beroperasi pada blok 256-bit dengan kunci 256-bit. Terdapat 32 Putaran (*round*) yang dikontstruksi dalam sebuah jaringan Feistel. Round Function yang digunakan adalah Round Function dari Rijndael dan Serpent yang masing-masing digunakan 16 kali. Penggunaan round function dilakukan secara selang-seling untuk setiap satu putaran jaringan Feistel. Sehingga terdapat spesfikasi turunan yaitu:

1. RoundKey untuk masing-masing algoritma Rijndael dan Serpent di-generate sejumlah 16 buah. (Berbeda dengan Rijndael original sejumlah 10 dan Serpent original sejumlah 32). Proses generasi RoundKey sama dengan algoritma yang original.
2. 256-bit blok masuk ke jaringan Feistel dan terbagi menjadi 2 yaitu L0 dan R0 , yang masing masing berukuran 128-bit.
3. Pertama kali R0 masuk ke RoundFunction pertama yaitu yang diambil dari algoritma Rjindael. Selanjutnya keluaran dari proses tersebut akan di XOR dengan L0. Proses ini merupakan satu putaran jaringan Feistel.
4. Pada putaran selanjutnya, proses yang dilakukan sama dengan c namun menggunakan RoundFunction dari algoritma Serpent.
5. Proses c dan d berlangsung selang-seling sampai total 32 putaran dan menghasilkan L16 dan R16

Untuk melakukan dekripsi, proses yang berlangsung serupa, hanya membalikkan dari putaran akhir jaringan Feistel



Gambar III-1 Konstruksi Algoritma dalam Jaringan Feistel

Dengan KR adalah Key Rijndael, R adalah Round Function Rjindael, KS adalah Key Serpent, dan S adalah Round Function Serpent.

IV. EKSPERIMEN DAN PEMBAHASAN HASIL

Algoritma yang dikembangkan pada makalah ini menggabungkan dua algoritma blok cipher lain yaitu Rijndael dan Serpent. Maka dari itu, untuk eksperimen akan dilakukan pembandingan dengan algoritma Rijnadael dan Serpent. Eksperiman dilakukan pada tiga mode yaitu ECB, CBC, dan

CFB 8-bit. Khusus untuk algoritma Serpent, panjang blok hanya mungkin 128 bit.

A. Mode Operasi ECB

1. Enkripsi

Tabel IV-1

Algoritma	zip file (11kB)
HARS	**40367653** ns 2910320 bytes
Serpent	42944299 ns **2851960** bytes
Rijndael	54502432 ns 2987176 bytes

Untuk enkripsi mode ECB Algoritma HARS unggul dalam runtime. Sedangkan konsumsi memori yang dipakai standar, hampir sama dengan dua algoritma lainnya.

2. Dekripsi

Tabel IV-2

Algoritma	zip file (11kB)
HARS	60829727 ns **2885416** bytes
Serpent	87801554 ns 2886104 bytes
Rijndael	**55296123** ns 2887832 bytes

Untuk dekripsi mode ECB Algoritma HARS menempati posisi tengah dalam runtime. Sedangkan konsumsi memori yang ketiga algoritma memiliki waktu yang hampir sama.

B. Mode Operasi CBC

1. Enkripsi

Tabel IV-3

Algoritma	zip file e (11kB)
HARS	40750244 ns **2884504** bytes
Serpent	63780412 ns 2885312 bytes
Rijndael	**37714602** ns 2912632 bytes

Pada mode CBC, algoritma HARS unggul dalam konsumsi memori.

2. Dekripsi

Tabel IV-4

Algoritma	zip file (11kB)
HARS	57735926 ns 2885232 bytes
Serpent	79273932 ns 2894712 bytes
Rijndael	**55778497** ns **2617304** bytes

Pada mode CBC, algoritma HARS hanya menempati posisi tengah. Baik runtime dan konsumsi memori dimenangkan oleh Rijndael.

C. Mode Operasi CFB 8-bit

1. Enkripsi

Tabel IV-4

Algoritma	zip file e (11kB)
HARS	383418575 ns 2908320 bytes
Serpent	**183333049** ns **2885312** bytes
Rijndael	644288214 ns 2906576 bytes

Pada mode CFB 8-bit ini, algoritma HARS hanya menempati posisi tengah. Baik runtime dan konsumsi memori dimenangkan oleh Serpent., namun algoritma HARS unggul runtime cukup jauh dibanding Rijndael.

2. Dekripsi

Tabel IV-5

Algoritma	zip file (11kB)
HARS	387130452 ns **2884872** bytes
Serpent	**207668931** ns 2926888 bytes
Rijndael	661865745 ns 2928272 bytes

Pada mode CFB 8-bit, algoritma HARS unggul dalam konsumsi memori.

V. ANALISIS KEAMANAN

A. Confusion

Pada dasarnya prinsip *confusion* adalah menghindari sedikitpun hubungan antara plainteks, cipherteks, dan kunci. Tujuannya adalah membuat kriptanalis frustasi untuk mencari pola statistik keterhubungan. Pada algoritma ini, diimplemantasikan substitusi dengan S-box yang digunakan oleh algoritma Rijndael dan Serpent. Operasi substitusi merupakan salah satu operasi untuk membantuk aspek confusion.

B. *Diffusion*

Prinsip Diffusion mengusahakan agar perubahan kecil pada plainteks (misal satu atau dua bit) dapat menghasilkan perubahan besar pada cipherteks, sehingga sulit untuk diprediksi. Dalam algoritma ini, digunakan cipher berulang, yaitu fungsi untuk setiap blok bit dilakukan sejumlah kali (round). Selain itu diluar dari agloritma block cipher, terdapat teknik lain untuk menimbulkan efek diffussion, yaitu penggunaan mode operasi CBC dan CFB yang dapat dipilih dalam algoritma ini.

VI. KESIMPULAN DAN SARAN

Dari eksperimen yang telah dilakukan, dapat dilihat bahwa kinerja HARS baik dari sisi kompleksitas waktu dan kompleksitas ruang, bernilai kurang lebih sama dengan kinerja Rijndael dan Serpent. Artinya perubahan-perubahan yang dilakukan pada HARS tidak mengurangi kinerja asli dari kedua Algoritma Rijndael dan Serpent yang digunakan sebagai dasar. Namun di sisi lain, penambahan struktur jaringan feistel dengan fungsi putaran yang diselang-seling antara Rijndael dan Serpent, memberikan lapisan keamanan tambahan yang membuat Block Cipher ini semakin sulit untuk dipecahkan. Lapisan kemanan tambahan ini tentu memberikan kepercayaan diri yang lebih atas kerahasiaan data yang di enkripsi menggunakan HARS. Ditambah lagi semakin meningkatnya kemampuan komputasi perangkat keras mengakibatkan tingkat keamanan yang tinggi menjadi suatu hal yang sangat krusial.

Untuk pengembangan selanjutnya, algoritma ini dapat dimodifikasi untuk mendukung besar blok yang bervariasi, seperti 128 bit dan 192 bit. Juga dapat dibuat pembagian blok kiri dan kanan pada jaringan feistel menjadi tidak selalu sama besar, sehingga tingkat keteracakan hasil enkripsi semakin bertambah. Selain itu dapat juga dipertimbangkan untuk menggunakan dua buah key yang berbeda untuk masing-masing fungsi putaran Rijndael dan Serpent sehingga tingkat keamanan menjadi semakin bertambah.

ACKNOWLEDGMENT

Seiring dengan selesainya penulis dalam menyusun makalah ini, penulis hendak menyampaikan puji dan syukur atas rahmat Allah SWT yang telah memberikan kemudahan bagi penulis selama pengerjaan makalah ini. Selanjutnya penulis juga ingin mengucapkan terima kasih kepada Bapak Rinaldi selaku dosen matakuliah IF4020 Kriptografi, yang telah menyampaikan ilmu yang dapat kami manfaatkan selama pengerjaan makalah ini. Tak lupa kami juga mengucapkan terima kasih kepada seluruh teman-teman yang baik secara langsung ataupun tidak langsung turut andil dalam pengerjaan makalah ini.

REFERENSI

[1] Anderson, R., Biham, E., & Knudsen, L. (1998). Serpent: A Proposal for The Advanced Encryption Standard.

[2] Daemen J. and Rijmen V. (1998). AES Proposal: Rijndael.

[3] Backes Michael, Lecture Notes for CS-578 Cryptography, Saarland University.

[4] Rinaldi. Bahan KuliahIF3058 Kriptografi: Algoritma Kriptografi Modern (Bagian 2)