Regina: Sebuah *Block Cipher* Baru Berdasarkan AES

Muhammad Garebaldhie Er Rahman
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13520029@std.stei.itb.ac.id

Frederik Imanuel Louis
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13520163@std.stei.itb.ac.id

Raden Rifqi Rahman
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13520166@std.stei.itb.ac.id

***Abstract*—Regina merupakan sebuah algoritma *block cipher* yang didasari pada beberapa algoritma *block cipher* lainnya, terutama AES. Regina memiliki ukuran *block* dan kunci yang sama yaitu 128 bit. Regina tersusun atas jaringan Feistel dengan 16 putaran. Beberapa komponen *cipher* Regina didasari pada komponen-komponen AES. Terdapat beberapa modifikasi komponen AES sebagai penyesuaian terhadap masukan yang diterima Regina. Algoritma Regina memiliki efek longsoran yang cukup baik ketika terdapat perbedaan satu bit pada *block* masukan. Regina juga cukup tahan dari serangan *bruteforce* karena memiliki ruang kunci yang cukup besar.**

Keywords—Kriptografi, Block Cipher, AES, DES, Feistel Network, Regina

I. PENDAHULUAN

Pada dunia digital, kebutuhan akan komunikasi dan transmisi data semakin meningkat. Namun, seiring meningkatnya kebutuhan akan komunikasi dan transmisi data, diperlukan suatu mekanisme ataupun cara untuk berkomunikasi dengan aman sehingga pesan yang dikirim tidak disalahgunakan oleh pihak-pihak yang tidak berwenang.

Sejak jaman dahulu, pada saat orang orang masih berkomunikasi dengan menggunakan surat, sudah terdapat konsep ataupun mekanisme untuk menjaga kerahasiaan pesan agar tetap aman yaitu ketika *Julius Caesar* menggunakan rotor untuk memutar pesan pada saat masa perang. Pada saat itu hal yang dilakukan *Julius Caesar* memiliki peranan penting dalam menyembunyikan informasi-informasi penting agar tidak disalahgunakan oleh pihak lawan ataupun yang tidak diinginkan. Pada dasarnya hal yang dilakukan Julius Caesar termasuk ke dalam kriptografi.

Kriptografi: ilmu dan seni untuk menjaga keamanan pesan. (Schneier, 1996). Kata *cryptography* berasal dari bahasa Yunani yaitu *cryptós* yang berarti tersembunyi dan *gráphein* yang berarti tulisan. Secara bahasa, kriptografi dapat diartikan sebagai *hidden writing* atau tulisan rahasia. Meskipun pada praktiknya saat ini, ilmu kriptografi tidak terbatas hanya pada tulisan saja namun bisa saja untuk media lainnya seperti gambar, audio serta video.

Pada kriptografi, jika suatu hal ingin dikatakan "aman," maka harus memenuhi empat sifat ataupun layanan: *confidentiality*, *data integrity*, *authentication*, serta *non repudiation. Confidentiality* artinya data yang kita kirimkan terjaga kerahasiaannya. *Data integrity* yaitu memastikan bahwa data yang kita kirim terjaga kerahasiaanya. *Authentication* berarti memastikan keaslian pengirim dan penerima, sedangkan *non-repudiation* yaitu pengirim ataupun penerima tidak dapat menyangkal jika sudah mengirim ataupun menerima pesan.

Karena perkembangan zaman, tantangan untuk memenuhi keempat layanan tersebut semakin meningkat secara drastis. Seiring perkembangan zaman, ilmu kriptografi semakin berkembang, mulai dari munculnya *symmetric key cryptography* yaitu kunci yang digunakan untuk enrkipsi dan dekripsi adalah kunci yang sama hingga *asymmetric key cryptography*. Selain itu, banyak bermunculan *cipher-cipher* baru mulai dari *classical cipher*, *modern cipher* berupa *stream cipher* serta *block cipher* ataupun steganografi untuk metode *digital watermarking*.

Untuk menjawab permasalahan diatas maka dibuatlah sebuah algoritma baru yaitu Regina. Regina merupakan aplikasi dari algoritma *block cipher* yang memanfaatkan jaringan Feistel. Regina memiliki beberapa aspek yang mirip dengan AES, yakni *S-Box* dan algoritma penjadwalan kunci '*key scheduling*'. Akan tetapi, terdapat sedikit modifikasi terhadap algoritma *key scheduling* AES untuk menyesuaikan panjang kunci putaran '*round key*'. Berdasarkan kesamaan ini, dirancang *block cipher* dengan fungsi putaran yang berbeda.

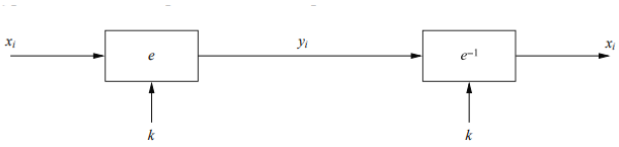
II. DASAR TEORI

A. Block Cipher

Block cipher adalah algoritma enkripsi yang bekerja pada suatu pesan yang akan dibagi menjadi blok-blok dengan ukuran yang sama. Enkripsi dilakukan pada masing-masing blok dengan menerima kunci. Dekripsi dilakukan dengan menerima *ciphertext* dan kunci. Kunci yang digunakan pada enkripsi dan dekripsi adalah kunci yang sama.

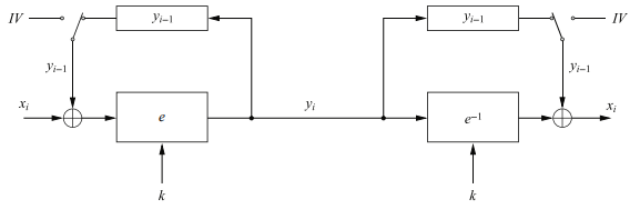
B. Mode Block Cipher

Block cipher dapat dilakukan dengan berbagai mode. Pertama, mode *Electronic Code Book* (ECB) melakukan enkripsi dan dekripsi pada tiap blok *plaintext* secara independen. Mode tersebut kurang bagus sebab tidak menghasilkan *avalanche effect.*



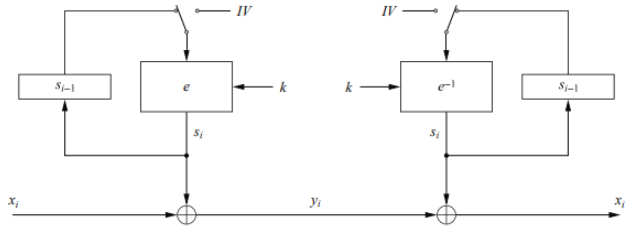
Gambar 1. Enkripsi dan dekripsi mode ECB. [1]

Kemudian, terdapat mode *Cipher Block Chaining* (CBC) mengambil output dari hasil enkripsi *block* sebelumnya, dan melakukan operasi xor nilai tersebut dengan *plaintext* blok berikutnya sebelum dienkripsi. Karena itu, mode ini juga membutuhkan suatu nilai *Initialization Vector* (IV) yang akan di-xor dengan blok *plaintext* pertama.



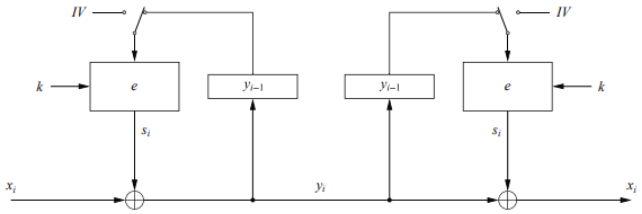
Gambar 2. Enkripsi dan dekripsi mode CBC. [1]

Mode *Output Feedback* (OFB) tidak melakukan enkripsi *plaintext* secara langsung. Untuk blok pertama, mode ini mengenkripsi IV, dan melakukan xor *plaintext* dan hasil enkripsi IV untuk memperoleh *ciphertext*. Kemudian, untuk blok-blok berikutnya, mode ini mengenkripsi hasil enkripsi dari IV tersebut dan melakukan operasi xor yang serupa.



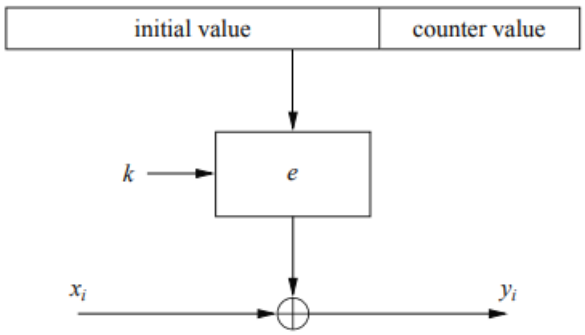
Gambar 3. Enkripsi dan dekripsi mode OFB. [1]

Mode *Cipher Feedback* (CFB) juga tidak melakukan enkripsi *plaintext* secara langsung. Untuk blok pertama, mode ini melakukan enkripsi terhadap IV, dan menerapkan operasi xor terhadap *plaintext* dan hasil enkripsi IV untuk mendapatkan *ciphertext*. Untuk blok-blok berikutnya, dilakukan enkripsi terhadap *ciphertext* yang dihasilkan pada blok sebelumnya, kemudian dilakukan operasi xor yang serupa.



Gambar 4. Enkripsi dan dekripsi mode CFB. [1]

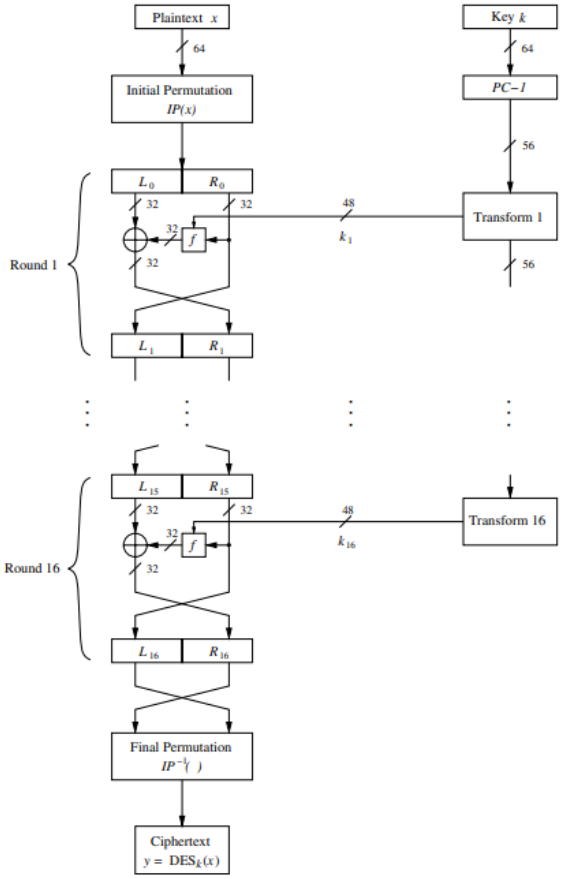
Mode *Counter* (CTR) menerima IV yang berukuran setengah ukuran blok, kemudian untuk setiap blok *plaintext*, membubuhkan IV dengan suatu *counter* (misalnya indeks blok), kemudian mengenkripsi IV yang sudah dibubuhkan tersebut. Hasil enkripsi akan di-xor dengan *plaintext* untuk menghasilkan *ciphertext*. Akibat konstruksi tersebut, enkripsi dan dekripsi pada mode CTR persis sama.



Gambar 5. Enkripsi dan dekripsi mode CTR. [1]

C. *Jaringan Feistel*

Jaringan Feistel merupakan suatu jaringan yang melakukan enkripsi blok dalam serangkaian ronde yang pada setiap rondenya dilakukan enkripsi sebagian *plaintext* pada suatu *round function* yang menerima *round key*. *Round key* dihasilkan dari suatu algoritma *key scheduling* yang menerima masukan *key* awal, dan menghasilkan berbagai *round key*. Jaringan Feistel menjamin bahwa struktur yang dihasilkan *reversible*, meskipun *round function* dan/atau *key scheduling* tidak *reversible*.



Gambar 6. Struktur jaringan Feistel. [1]

III. RANCANGAN BLOCK CIPHER

Block cipher Regina menggunakan jaringan Feistel seimbang dengan 16 ronde dengan ukuran blok 128 bit dan ukuran kunci 128 bit. Pada setiap iterasi jaringan Feistel, *state* 128 bit akan dibagi menjadi dua bagian, dengan masing-masing 64 bit. Salah satu bagian tersebut akan dimasukkan ke dalam *round function* bersama round key yang akan menghasilkan *output* 64 bit. Kemudian, hasil dari *round function* akan di-xor dengan bagian *state* lainnya yang tidak dimasukkan ke *round function*. Terakhir, hasil xor akan disambung dengan paruh *state* yang dimasukkan ke *round function* untuk menjadi *state* pada iterasi berikutnya (berukuran 128 bit).

A. *Key Scheduling*

Key scheduling pada Regina dirancang dengan algoritma yang serupa dengan *key scheduling* pada AES-128. Regina menerima kunci eksternal sepanjang 128 bit yang kemudian

digunakan untuk membangkitkan *round key* secara *word* per *word* dengan panjang *word* sebesar 32 bit.

Berdasarkan referensi [3], kunci eksternal sepanjang 128 bit dalam bentuk $K_1K_2K_3K_4$ dipecah ke dalam 4 *word* menjadi $W_1 = K_1$, $W_2 = K_2$, $W_3 = K_3$, dan $W_4 = K_4$. Selanjutnya, untuk $i \geq 4$, *word* kunci dibangkitkan menggunakan fungsi

$$W_i = \begin{cases} W_{i-4} \oplus Sub(Rot(W_{i-1})), & i \equiv 0 \ (mod\ 4) \\ W_{i-4} \oplus W_{i-1}, & i \not\equiv 0 \ (mod\ 4) \end{cases}$$

dengan *Rot* dan *Sub* adalah fungsi yang dihitung sebagai berikut.

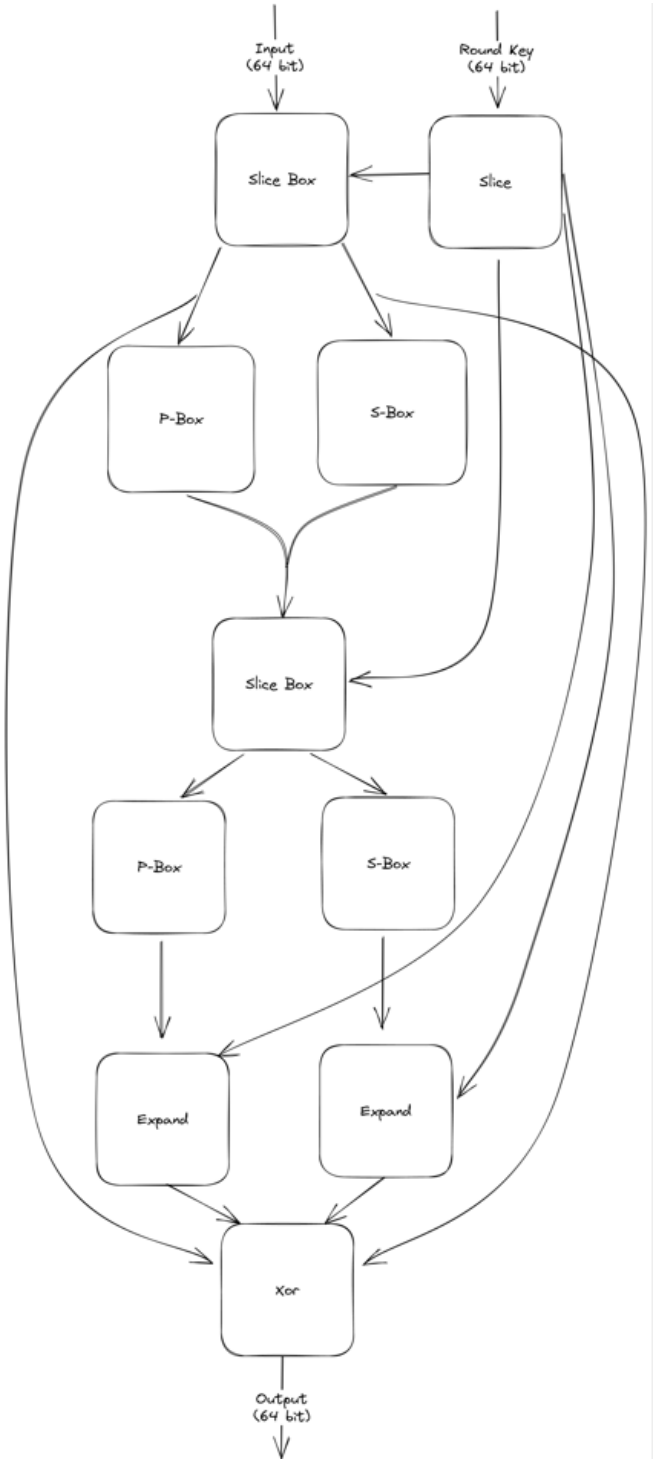
$$Rot(W) = Rot(b_0b_1b_2b_3) = b_1b_2b_3b_0$$

$$Sub(W) = Sub(b_0b_1b_2b_3)$$
$$= SBox(b_0)SBox(b_1)SBox(b_2)SBox(b_3)$$

Karena *round key* yang diperlukan dalam *round function* adalah 64 bit, maka kunci yang dihasilkan adalah pasangan *word* kunci yang dibangkitkan berdasarkan algoritma di atas. Akan tetapi, 32 *word* pertama—W_1 hingga W_{32}—yang dibangkitkan akan dibuang sehingga 16 kunci yang digunakan pada *round function* berasal dari W_{33} hingga W_{64}. Hal ini dilakukan agar *round key* yang dihasilkan semakin sulit ditebak sehingga meningkatkan keamanan.

B. Round Function

Round function akan menerima masukan paruh *block* (*input*) 64 bit dan *round key* 64 bit, dan menghasilkan *output* 64 bit. *Round function* terdiri atas berbagai komponen, seperti *Slice box*, *S-Box*, *P-Box*, *Expand*, xor, dan *slice*. Pertama, *round key* akan dibagi menjadi 4 bagian, masing-masing 16 bit untuk digunakan pada komponen yang berbeda. Kemudian, *input* 64 bit akan dibagi menjadi dua, masing-masing 32 bit, menggunakan *Slice box*, dengan masukan *round key* bagian pertama berukuran 16 bit. Kemudian, salah satu dari *state* 32 bit tersebut akan dimasukkan kedalam *S-Box*, dan yang lainnya akan dimasukkan pada *P-Box*. Output dari kedua komponen tersebut akan dikonkatenasi, dan dimasukkan lagi ke dalam *Slice box*, sehingga kembali dihasilkan dua state 32 bit. Kemudian, salah satu state akan dimasukkan ke *S-Box*, dan yang lainnya pada *P-Box*. *Output* dari kedua komponen tersebut akan dimasukkan kedalam komponen *Expand*, menghasilkan dua output 64 bit. Kemudian, kedua output ini akan di-xor, bersama dengan kedua output dari *Slice box* yang pertama. Hasil xor dari 4 *state* tersebut akan menghasilkan *output* 64 bit yang diinginkan.



Gambar 7. Skema *round function* Regina.

C. Slice Box

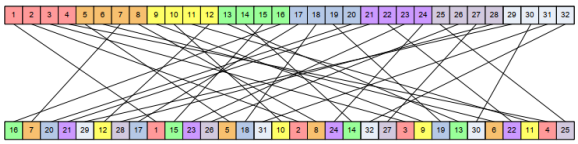
Slice box akan menerima satu *input* 64 bit dan menghasilkan *dua output* 32 bit, dengan menggunakan kunci 16 bit. *Output* akan dibentuk menggunakan *Linear Congruental Generator* (LCG), dimana akan dipilih bit-bit dari *input* 64 bit untuk menjadi bit-bit pada *output*. *State* dari LCG sendiri dibentuk dari *key* masukan. Berikut implementasi dari *Slice box* dalam Bahasa Python.

```
   def _slice_box(self, input: bytes, key:
bytes) -> (bytes, bytes):

       assert(len(key) == 2)
```

```
        assert(len(input) == 8)


        key_binary = format(int.from_bytes(key,
"big"), '#018b')[2:]
        n = 64
        a = int(key_binary[:6], 2)
        init = int(key_binary[6:12], 2)
        cnt = int(key_binary[12:], 2)


        a = a*3//gcd(a, n)
        init = (init + cnt * a) % 64


        left = ""
        right = ""
        input_binary =
format(int.from_bytes(input, "big"),
'#066b')[2:]
        for _ in range(32):
            left += input_binary[init]
            init = (init + a) % 64
            right += input_binary[init]
            init = (init + a) % 64
        left = int(left, 2).to_bytes(4, "big")
        right = int(right, 2).to_bytes(4, "big")
    return (left, right)
```

D. *P-Box*

P-Box pada Regina dibangkitkan secara statik dengan panjang box yaitu 32 buah. Angka angka yang ada pada P-Box merepresentasikan posisi bit setelah dilakukan permutasi.



Gambar 8. Ilustrasi *P-Box*. [5]

Bagian atas merepresentasikan posisi input sebelum dilakukan permutasi. *P-Box* direpresentasikan pada gambar bagian bawah. *P-Box* direpresentasikan dengan bilangan berurutan seperti pada barisan bilangan pada kotak yang bawah. Angka 16 pada bagian bawah artinya, bit pada posisi ke-16 akan berada pada urutan ke-0.

Berikut P-Box yang digunakan pada Regina.

TABEL I. P-BOX REGINA

14	28	9	18	19	20	27	26
17	31	23	11	5	12	15	4
6	21	25	8	13	0	30	2
29	7	22	3	1	10	16	24

E. *S-Box*

S-Box pada Regina juga dirancang berdasarkan *S-Box* AES. Penggunaan *S-Box* ini dipilih karena dapat memberikan hasil yang cukup baik pada algoritma *key scheduling* yang dirancang berdasarkan eksperimen. Berdasarkan referensi [4], berikut adalah *S-Box* yang digunakan.

TABEL II. S-BOX AES

Nibble **Pertama**	***Nibble* kedua**							
	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***
0	63	7c	77	7b	f2	6b	6f	C5
1	ca	82	c9	7d	fa	59	47	f0
2	b7	fd	93	26	36	3f	f7	cc
3	04	c7	23	c3	18	96	05	9a
4	09	83	2c	1a	1b	6e	5a	a0
5	53	d1	00	ed	20	fc	b1	5b
6	d0	ef	aa	fb	43	4d	33	85
7	51	a3	40	8f	92	9d	38	f5
8	cd	0c	13	ec	5f	97	44	17
9	60	81	4f	dc	22	2a	90	88
a	e0	32	3a	0a	49	06	24	5c
b	e7	c8	37	6d	8d	d5	4e	a9
c	ba	78	25	2e	1c	a6	b4	c6
d	70	3e	b5	66	48	03	f6	0e
e	e1	f8	98	11	69	d9	8e	94
f	8c	a1	89	0d	bf	e6	42	68
	***Nibble* kedua**							
	8	***9***	***a***	***b***	***c***	***d***	***e***	***f***
0	30	01	67	2b	fe	d7	ab	76
1	ad	d4	a2	af	9c	a4	72	c0
2	34	a5	e5	f1	71	d8	31	15
3	07	12	80	e2	eb	27	b2	75
4	52	3b	d6	b3	29	e3	2f	84
5	6a	cb	be	39	4a	4c	58	cf
6	45	f9	02	7f	50	3c	9f	a8
7	bc	b6	da	21	10	ff	f3	d2
8	c4	a7	7e	3d	64	5d	19	73
9	46	ee	b8	14	de	5e	0b	db
a	c2	d3	ac	62	91	95	e4	79
b	6c	56	f4	ea	65	7a	ae	08
c	e8	dd	74	1f	4b	bd	8b	8a
d	61	35	57	b9	86	c1	1d	9e
e	9b	1e	87	e9	ce	55	28	df
f	41	99	2d	0f	b0	54	bb	16

F. *Expand*

Setelah dilakukan operasi operasi seperti *P-Box* dan *S-Box*, akan dilakukan *expand* karena *output* dari proses sebelumnya adalah 32 bit. *Input* tersebut akan di-*expand* menjadi 64-bit *output*. Operasi ini menerima dua parameter yaitu *input* yang memiliki Panjang 32 bit dan *key* yang memiliki panjang 16 bit. Pada proses ekspansi ini terdapat beberapa jenis operasi yang dilakukan yang ditandai dengan angka 0, 1, dan 2.

- Operasi 0 menandakan bahwa akan diambil bit dari bagian input mulai dari paling kanan.
- Operasi 1 menandakan bahwa akan diambil bit dari bagian *key* mulai dari paling kanan.
- Operasi 2 menandakan bahwa akan diambil *random bytes* berdasarkan jumlah iterasi lalu di-xor dengan *key* dan lalu diambil bit paling kanan.

Algoritma ekspansi akan bekerja sebagai berikut

1. Lakukan *loop* sebanyak 64 kali.
    a. Ambil jenis operasi yang ada pada table operasi mulai dari indeks ke-0.
    b. Lakukan operasi sesuai definisi di bawah ini.
    c. Jika operasi 2 yang terpilih:
        i. Lakukan penambahan nilai *key* dalam representasi *integer* dengan jumlah iterasi pada *loop*.
        ii. Lakukan mod 32 dan simpan sebagai variable rand_num.
        iii. Lakukan xor *key* dengan *input* yang *shift left* sebesar rand_num
2. Semua langkah tersebut akan menghasilkan 64 bit baru.
3. Ubah bit tersebut ke dalam representasi *bytes*.

$$O_i = \begin{cases} (input \gg input\_shift\_count)\ \&\ 1, & i = 0 \\ (key \gg key\_shift\_count)\ \&\ 1, & i = 1 \\ ((key) \oplus (input \ll rand\_num))\ \&\ 1, & i = 2 \end{cases}$$

rand_num diambil dari representasi *integer key* ditambah dengan jumlah iterasi yang nantinya akan dilakukan modulo 32.

```
rand_num = (key_int + i) mod (32)
```

Berikut merupakan pilihan operasi yang telah didefinisikan secara statis.

TABEL III. OPERASI SEKUENSIAL YANG DILAKUKAN

1	2	1	0	0	0	0	0
2	0	2	0	1	1	0	2
1	1	0	0	0	1	2	2
1	2	0	0	2	2	2	0
2	0	0	1	0	0	1	0
0	0	0	1	2	1	0	1
0	0	0	1	2	0	2	1
0	2	2	0	0	0	0	1

IV. EKSPERIMEN DAN PEMBAHASAN HASIL

A. Waktu Enkripsi dan Dekripsi

Berdasarkan rancangan yang diajukan, dilakukan implementasi Regina dalam bahasa Python. Selanjutnya, implementasi Regina diuji dengan cara melakukan enkripsi dan dekripsi 4 *file* yang masing-masing merupakan *file* gambar (PNG), *file* dokumen (PDF), *file* arsip (ZIP), dan *file* media (MP4). Berikut adalah waktu enkripsi dan dekripsi untuk keempat *file* tersebut.

1. Format: png
   Ukuran: 22.98 KB
   Waktu enkripsi: 12.154 detik
   Waktu dekripsi: 12.393 detik
2. Format: pdf
   Ukuran *file*: 285.86 KB
   Waktu enkripsi: 143.205 detik
   Waktu dekripsi: 141.109 detik
3. Format: zip
   Ukuran: 817.2 KB
   Waktu enkripsi: 412.74 detik
   Waktu dekripsi: 407.816 detik
4. Format: mp4
   Ukuran: 2.97 MB
   Waktu enkripsi: 1497.611 detik
   Waktu dekripsi: 1557.379 detik

Berdasarkan hasil pengujian, diperoleh waktu enkripsi dan dekripsi *file* yang lambat meskipun ukuran *file* cukup kecil. Setelah meninjau kembali implementasi *cipher* yang dibuat, selain karena implementasi yang dilakukan dalam bahasa Python, lambatnya enkripsi dan dekripsi yang dilakukan diperkirakan disebabkan oleh implementasi fungsi ekspansi (*expand*) pada *round function* yang mengoperasikan masukan secara bit-per-bit. Akibatnya, terjadi sangat banyak operasi bit pada fungsi tersebut yang menyebabkan komputasi enkripsi dan dekripsi menjadi lambat.

B. Analisis Efek Longsoran

Selain pengujian dalam aspek waktu enkripsi dan dekripsi, implementasi Regina juga diuji dalam aspek efek longsoran. Pengujian dilakukan dengan melakukan enkripsi terhadap satu *block* masukan acak '*random*' kemudian membandingkannya dengan hasil enkripsi *block* tersebut, tetapi memiliki satu bit yang dibalik yang juga dipilih secara acak. Berdasarkan hasil pengujian, Regina memiliki efek longsoran yang cukup baik. Sebagai contoh, berikut adalah beberapa perbandingan hasil enkripsi *block* dengan satu bit berbeda.

TABEL IV. PERBANDINGAN HASIL ENKRIPSI BLOK DENGAN SATU BIT BERBEDA

Block	***Hex String***	***Hex String* Enkripsi**	**Perbedaan**
Asli	1001bf8f7edacc48 9ad7dc7e5aec95e5	7a8fe9de5514274a c9d091f2b7c2e4ed	16/16 byte 70/128 bit
Serupa	1001bf8f7eda4c48 9ad7dc7e5aec95e5	d6f44a8a2c8fe5ab 908efa5147b6dbc0	
Asli	9cb9372525cee85f 4f9555e043101c68	e91b9cb617f8ed47 05f308d19fc683c1	16/16 byte 65/128 bit
Serupa	9cb9372525cee85f 4f9555e043901c68	bd906db03311915f fb2271057947ba34	
Asli	e130774b19385a7f 6fbba1b5dfe1bf4c	609885337f4289e0 ebeb459f8bc1323c	16/16 byte 67/128 bit
Serupa	e130674b19385a7f 6fbba1b5dfe1bf4c	7fba5f6d9bca338f 40fde842aaa5f767	

Berdasarkan contoh di atas, diperoleh hasil yang cukup baik, yakni hasil enkripsi *block* asli dengan *block* yang memiliki satu bit berbeda memiliki perbedaan *byte* sebesar 100%. Perbedaan bit dari hasil tersebut juga menunjukkan angka yang cukup baik yang berada di kisaran 50%.

Dengan melakukan pengujian ini secara berulang-ulang, diperoleh hasil yang cukup konsisten, yakni setiap hasil enkripsi selalu memiliki perbedaan *byte* sebanyak 15-16 *byte* atau 93.75-100% dan perbedaan bit sekitar 40-60%.

C. Besar Ruang Kunci dan Analisis Serangan Bruteforce

Berdasarkan rancangan yang diajukan, Regina menerima masukan kunci sepanjang 128 bit. Untuk masukan kunci dengan panjang kurang dari 128 bit, implementasi Regina akan mengulang karakter-karakter pada kunci masukan secara berurutan hingga diperoleh kunci sepanjang 128 bit. Untuk kunci dengan panjang lebih dari 128 bit, bit-bit setelah bit ke-128 akan dipangkas sehingga kunci yang digunakan adalah

128 bit pertama. Dengan demikian, karena panjang kunci selalu tetap, yaitu sepanjang 128 bit, maka ruang kunci memiliki ukuran sebesar $2^{128} \approx 3.403 \cdot 10^{38}$.

Dengan besarnya ruang kunci Regina, serangan *bruteforce* terhadap hasil enkripsi Regina akan memerlukan waktu yang lama. Jika dapat dilakukan 10^6 kali enkripsi setiap detik, maka dibutuhkan waktu selama $3.403 \cdot 10^{32}$ detik atau $1.079 \cdot 10^{25}$ tahun untuk memeriksa semua kunci yang mungkin. Oleh karena itu, Regina cukup aman terhadap serangan *bruteforce*.

V. KESIMPULAN DAN SARAN

Regina merupakan sebuah algoritma *block cipher* yang disusun menggunakan jaringan Feistel. Regina membagi masukan biner ke dalam *block* sepanjang 128 bit. Regina melakukan 16 putaran pada jaringan Feistel untuk setiap *block* tersebut. Regina juga menerima masukan kunci sepanjang bit 128 bit.

Algoritma Regina cukup tidak efisien dalam melakukan enkripsi dan dekripsi. Untuk masukan berukuran kurang dari 1 MB, enkripsi dan dekripsi dengan Regina dapat memerlukan waktu hingga beberapa menit. Hal ini diakibatkan oleh rancangan fungsi ekspansi dalam fungsi putaran yang mengoperasikan masukan bit-per-bit. Akan tetapi, Regina memiliki efek longsoran yang cukup baik ketika terjadi perbedaan 1 bit dalam *block* masukan. Regina juga cukup tahan dari serangan *bruteforce* karena kunci yang cukup panjang.

Meninjau kekurangan Regina pada lamanya waktu enkripsi dan dekripsi, terdapat ruang untuk memperbaiki Regina agar memiliki performa yang lebih baik. Untuk meningkatkan efisiensi Regina, fungsi ekspansi pada *round function* dapat disederhanakan sehingga dapat mengoperasikan masukan secara *byte*-per-*byte*. Dengan demikian, diharapkan waktu enkripsi dan dekripsi Regina dapat menurun hingga mencapai angka yang layak.

VI. UCAPAN TERIMA KASIH

Pertama-tama, kami memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa yang telah memberikan kami nikmat yang tidak terhingga, pengetahuan, dan kesempatan untuk menyelesaikan makalah ini, kami juga mengucapkan terima kasih kepada pengajar IF4020, Dr. Ir. Rinaldi Munir, M.T., yang telah mendorong kami untuk menulis makalah ini.

REFERENSI

[1] Christof Paar and Jan Pelzl, Understanding Cryptography: A textbook for students and practitioners.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2022-2023/kripto22-23.htm, diakses pada 3 Maret 2023

[3] J. Daemen and V. Rijmen. (Sep. 3, 1999). AES proposal: Rijndael. [Online]. Available: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

[4] Advanced Encryption Standard, FIPS 197, National Institute of Standards and Technology, USA, Nov. 26, 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[5] R. Munir, "Perancangan Block Cipher," in Cryptography, diakses pada 3 Maret 2023.