Studi Algoritma KASUMI (A5/3) *Block Cipher*

Donnie - NIM: 13505606
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
E-mail: if15606@students.if.itb.ac.id

Abstrak

Makalah ini akan membahas mengenai algoritma KASUMI yang sekarang banyak digunakan untuk aplikasi-aplikasi yang menggunakan *Universal Mobile Telecomunications System* (UMTS). UMTS adalah sebuah standar internasional untuk sistem komunikasi *mobile* 3G. UMTS mengimplementasikan algoritma yang efisien, dan menyediakan saluran yang aman.

(*European Telecommunication Standards Institute*) ETSI mendeklarasikan algoritma KASUMI sebagai algoritma kriptografi standar untuk aplikasi-aplikasi UMTS. KASUMI adalah algoritma dasar untuk algoritma stream cipher f8 dan f9.

Standar komunikasi mobile, alogoritma A5/3, dibangun berdasarkan algoritma KASUMI. Algoritma ini dispesifikasi oleh *3rd Generation Partnership Project* (3GPP) untuk penggunaan sistem *mobile* untuk 3G sebagai algoritma inti untuk kerahasiaan dan integritas.

Algoritma A5/3 telah dikembangkan oleh *GSM Association Security Group* dan 3GPP untuk digunakan dalam sistem GSM. Algoritma tersebut juga akan dapat digunakan untuk *General Packet Radio Service* (GPRS), yang akan disebut sebagai GEA3, dan mode GSM lainnya seperti *High Speed Circuit Switched Data* (HSCSD) dan *Enhanced Data Rates for GSM Evolution* (EDGE).

Kata Kunci: *Universal Mobile Telecommunications System* (UMTS), *General Packet Radio Service* (GPRS), *High Speed Circuit Switched Data* (HSCSD), *Enhanced Data Rates for GSM Evolution* (EDGE).

1. Pendahuluan

Saat ini, komunikasi *mobile* sudah menyebar ke seluruh dunia dengan sangat cepat, dengan perkembangan teknologi yang tidak kalah cepat pula. Teknologi komunikasi *mobile* telah mencapai generasi ketiga (3G) dimana telah dapat melakukan transfer data suara dan data lainnya secara bersamaan.

Tetapi tentunya dalam proses komunikasi maupun transfer data tersebut memerlukan sistem keamanan yang dapat diandalkan, untuk menjaga kerahasiaan data dan komunikasi yang dilakukan oleh para pengguna [4].

Awalnya, untuk komunikasi berbasiskan GSM, algoritma enkripsi yang digunakan adalah algoritma A5/1 dan A5/2. Tetapi kedua algoritma ini tidak bertahan lama karena algoritma yang dirahasiakan dan ternyata memiliki banyak kelemahan yang membuatnya rentan terhadap beberapa jenis serangan. Selanjutnya, saat kedua algoritma tersebut dipecahkan melalui proses *reversed-engineering*, barulah diketahui banyaknya kelemahan yang dimiliki oleh kedua algoritma tersebut, sehingga keduanya dinilai tidak cocok lagi untuk digunakan dan harus segera dicari penggantinya [2][5].

Pada tahun 2002, sebuah algoritma keamanan baru, yang dikenal sebagai A5/3, telah memberikan tingkat keamanan yang lebih tinggi terhadap *eavesdropping*. Bahkan algoritma ini juga memasikan bahwa walaupun seseorang dapat mengambil sinyal percakapan GSM, ia tidak dapat memahami hasil yang ia curi tersebut, walaupun melalui proses komputasi yang tinggi [3].

Tetapi sebetulnya, algoritma A5/3 ini didasari oleh algoritma lain yang dikembangkan oleh Security Algrithms Group of Experts (SAGE), yang merupakan bagian dari European Telecommunications Standards Institute (ETSI), yaitu KASUMI. SAGE diminta untuk mengembangkan sebuah algoritma untuk *mobile telephone* generasi ketiga, berdasarkan spesifikasi yang diberikan oleh SA-WG2.

Akibat jadwal tenggat waktu yang singkat, maka SAGE memutuskan bahwa mereka akan menggunakan algoritma yang telah ada sebelumnya sebagai dasar dari algritma baru yang akan mereka kembangkan, yaitu

MISTY1. Alasan pemilihan algoritma ini tidak tanpa alasan yang baik, tetapi didasarkan karena MISTY1 memiliki tingkat keamanan yang tinggi, algoritma ini juga merupakn satu-satunya algoritma block cipher yang telah memenuhi spesifikasi dari 3GPP yang menyatakan bahwa algoritma tersebut dapat berjalan pada perangkat keras yang menggunakan tidak lebih dari sepuluh ribu gerbang logika.

Maka dilahirkanlah sebuah algoritma varian dari MISTY1, dengan 64-bit block cipher dan kunci 128-bit, yang diberi nama KASUMI, yang merupakan bahasa jepang untuk 'kabut' atau '*misty*' itu sendiri. Algoritma ini akhirnya diterima secara formal sebagai mandatory cipher dalam W-CDMA pada Januari 2000 [3].

2. Kebijakan Perancangan MISTY1

MISTY1 dirancang berdasarkan tiga buah prinsip:
- MISTY harus memiliki basis numerik untuk keamanannya.
- MISTY harus dapat bekerja dengan cepat pada sebuah aplikasi tanpa terikat kepada prosesor yang dimiliknya.
- MISTY harus dapat bekerja dengan cepat pada implementasi perangkat keras.

Algoritma KASUMI yang dirancang terbukti aman terhadap kriptanalisis linear dan diferensial. Hal ini terjadi karena algoritma tersebut dibangun dari komponen yang lebih kecil yang telah terbukti kebal terhadap kedua jenis serangan tersebut. Struktur Feistel dari MISTY1 dilakukan berulang secara rekursif dalam fungsi round FO dan kernel FI.

Pembagian yang tidak sama dari FI didasarkan kepada bahwa fungsi bijeksi pada ukuran blok ganjil lebih baik dibandingkan dengan yang memiliki ukuran genap. Hal ini dilihat dari sudut pandang yang telah terbukti keamanannya terhadap kriptanalisis linear dan diferensial.

Dalam penentuan S-box S7 dan S9, kriteria yang diikuti adalah sebagai berikut:
- Rata-rata probabilitas linear dan diferensial haruslah cukup rendah.
- Waktu jeda yang muncul pada implementasi perangkat keras haruslah secepat mungkin.
- Tingkat algebraic yang dimilikinya haruslah setinggi mungkin, jika memungkinkan.

Fungsi hasil ditemukan dengan mencari fungsi bentuk A (X^t) terhadap GF(2^7) dan GF(2^9), dimana A adalah fungsi transformasi linear bijektif. Derajat non-linear dari S7 adalah 3, sedangkan untuk S9 adalah 2.

Untuk menghindari kemungkinan terjadinya serangan selain dari kriptanalisis linear dan diferensial, maka rancangan MISTY1 dilengkapi dengan fungsi FL. Fungsi ini bersifat linear untuk panjang kunci yang tetap, tetapi memiliki bentuk yang bersifat variabel tergantung kepada nilai dari kunci trersebut.

Penjadwalan kunci dari MISTY1 dirancang dengan mengikuti prinsip sebagai berikut:
- Ukuran dari kunci sebesar 128 bit.
- Ukuran dari sub-kunci sebesar 256 bit.
- Setiap iterasi dipengaruhi oleh semua bit kunci.
- Sebanyak mungkin bit dari sub-kunci memiliki pengaruh pada setiap iterasi.

3. Perubahan dari MISTY1 ke KASUMI

Berikut adalah perubahan yang diterapkan terhadap MISTY1 saat merancang KASUMI.

- Data Enkripsi

a) Mengubah lokasi dari fungsi FL.
Hal ini membuat implementasi dari perangkat keras menjadi lebih sederhana, tetapi lebih lambat. Tetapi kelemahan ini dimitigasi dengan perubahan lainnya.

b) Menghilangkan subkunci KO_{i4} pada fungsi FO.
Hal ini membuat implementasi pada perangkat keras menjadi lebih cepat dan sederhana; sekarang fungsi FO memiliki struktur pengulangan yang sederhana.

c) Menambahkan fungsi pergeseran rotasi pada fungsi FL.
Diasumsikan bahwa hal ini membuat proses kriptanalisis lebih sulit; tidak ada dampak negatif terhadap ukuran maupun kecepatan perangkat keras.

d) Mengubah tabel substitusi dari S7.
Tidak ada perubahan yang berarti, hanya menyusun urutan dari bit sebelum dan sesudah S7 yang asli. Tidak ada tabel yang lebih baik jika dipandang dari sudut pandang impleemntasi peangkat keras.

e) Mengubah tabel substitusi S9.
Hal ini membuat perangkat keras menjadi lebih kecil dan lebih cepat. Jumlah total dari "term" dari S9 yang baru lebih kecil dibandingkan dengan yang asli.

f) Menambahkan S7 pada fungsi FI.

Hal ini akan sangat meningkatkan tingkat keamanan, tetapi membuat perangkat keras yang lebih besar. Diharapkan bahwa pembesaran ini akan dikompensasi oleh pengurangan penjadwalan kunci.

- Penjadwalan Kunci

a) Menghilangkan seluruh fungsi FI di bagian penjadwalan kunci.
Hal ini akan membuat perangkat keras yang lebih kecil dan/atau mengurangi waktu yang digunakan untuk mempersiapkan kunci. Diharapkan bahwa serangan yang berhubungan dengan kunci tidak akan bekerja untuk struktur ini.

b) Menambahkan nilai konstanta C_i dan operasi pergeseran rotasi bit.
Hal ini dapat menghindari penggunaan subkunci yang sama untuk iterasi yang berbeda.

4. 3GPP Algorithm Requirements

3GPP mensyaratkan adanya dua algoritma yang harus diimplementasikan dengan sempurna, yaitu algoritma f8 – *Confidential Algorithm*, dan f9 – *Integrity Algorithm* [].

Berikut adalah spesifikasi utama dari kedua algoritma tersebut:

f8 – Confidential Function

- Fungsi f8 hanya akan digunakan untuk melindungi kerahasiaan dari data yang dimiliki pengguna dan data yang dipancarkan melalui hubungan akses radio antara UE dan RNC.

- Algoritma tersebut harus dirancang untuk mengakomodasi sejumlah pilihan implementasi, termasuk perangkat keras dan lunak.

- Untuk implementasi perangkat keras, harus dapat mengimplementasikan algoritma tersebut dengan menggunakan kurang dari 10.000 gerbang logika.

- Implementasi dari algoritma tersebut harus dapat mencapai kecepatan enkripsi sebesar 2MBit/s pada saat *uplink* maupun pada saat *downlink*.

- Algoritma f8 akan digunakan untuk melakukan enkripsi *frame* dengan variabel panjang hingga 5000-bit.

- Fungsi f8 haruslah merupakan *symmetric synchronous stream cipher*.

- Panjang dari kunci **CK** adalah 128 bit. Jika panjang kunci harus dibuat lebih kecil dari 128 bit, maka MSB dari **CK** akan berisi informasi dari kunci efektif, sedangkan sisa LSB akan berisi bit 0.

- Parameter input tambahan: *COUNT, BEARER, DIRECTION and LENGTH.*

- Blok dari plainteks berisi payload dari RLC PDU / MAC SDU untuk dienkripsi pada sebuah frame sepanjang 10ms physical layer untuk arah bearer maupun transmission. Blok plainteks dapat berisi *user traffic* ataupun *signaling data*. Struktur dari blok plainteks tidak dapat dispesifikasikan pada saat ini.

Dalam proses implementasi pada KASUMI, fungsi f8 diterapkan dengan menggunakan stream cipher yang digunakan untuk enkripsi dari frame data, yang dibentuk dari KASUMI dengan bentuk varian dari *Output Feedback Mode* (OFB) dengan feedback sebesar 64-bit. Ilustrasi implementasi dari fungsi f8 dapat dilihat pada Gambar 1.

Saat fase pra-komputasi, parameter sistem *COUNT*, *BEARER*, dan *DIRECTION* ditambahkan padding untuk menjadi blok data yang memiliki panjang yang lengkap, kemudian dienkripsi dalam KASUMI dengan kunci turunan CK'. Kunci turunan CK' ini didapatkan dari hasil operasi XOR dari kunci CK yang asli dengan sebuah fixed mask. Keluaran dari proses ini yaitu sebuah nilai Register A sebesar 64-bit, yang merupakan masukan dari setiap perhitungan dari KASUMI.

Blok *keystream* berikutnya kemudian dibangkitkan dengan menjalankan KASUMI pada keluaran dengan tambahan masukan A dan sebuah *block counter* (BLKCTR) ke dalam *feedback*. Selanjutnya cipherteks dihasilkan sebagai hasil operasi XOR dari bit-bit keystream dengan bit dari plainteks.

f9 – Integrity Function

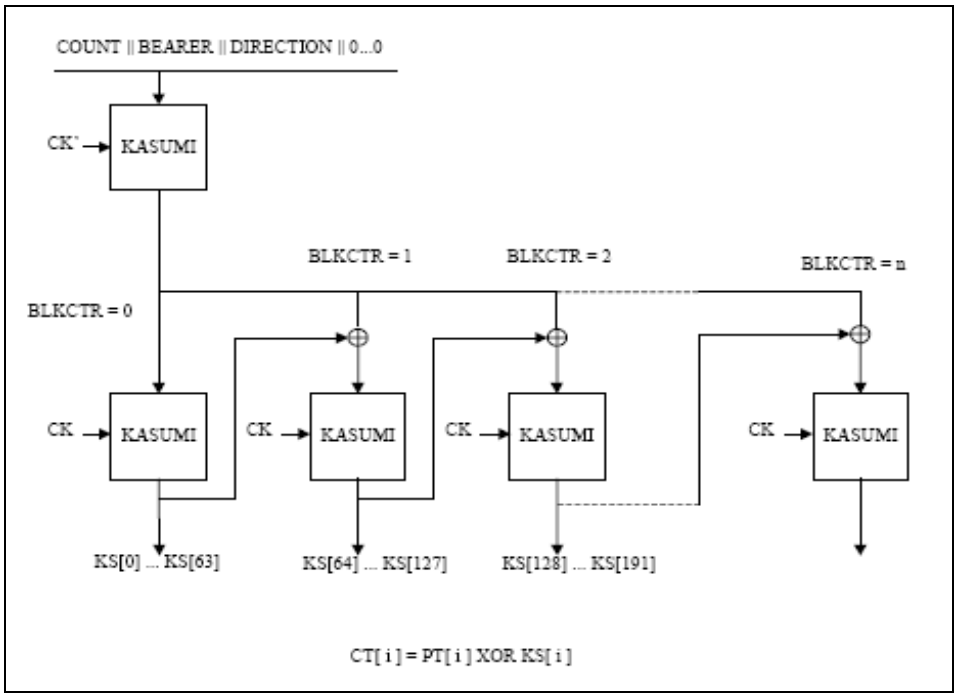
- Fungsi MAC f9 akan digunakan untuk melakukan otentikasi integritas dari data dan asal data yang dikirimkan yang ditransmisikan antara UE dan RNC.

- Algoritma tersebut harus dirancang untuk mengakomodasi sejumlah pilihan implementasi, termasuk perangkat keras dan lunak.

- Fungsi f9 akan menjadi fungsi MAC.

- Panjang dari kunci integritas **IK** adalah 128 bit. Jika panjang kunci harus dibuat lebih

kecil dari 128 bit, maka MSB dari **IK** akan berisi informasi dari kunci efektif, sedangkan sisa LSB akan berisi bit 0.
- Parameter input tambahan: *COUNT, FRESH and LENGTH.*
- Algortima tersebut akan menghasilkan keluaran 32-bit MAC.



Gambar 1 Fungsi f8 pada Algoritma KASUMI

Fungsi f9 melakukan proses komputasi terhadap sebuah 32-bit *Message Authentication Code* (MAC) pada sebuah pesan masukan dengan menggunakan kunci integritas IK. Panjang pesan dapat sepanjang 1 sampai dengan 5000 bit. Struktur dari fungsi f9 diilustrasikan pada Gambar 2.

Fungsi f9 adalah sebuah varian dari fungsi standar CBC MAC, yang didefinisikan dalam ISO 9797. Pada struktur ini, hasil XOR dari seluruh komputasi dari setiap blok di-XOR-kan untuk menjadi masukan dari proses komputasi akhir dari KASUMI. MAC pada pesan masukan berisi sebagian keluaran bagian kiri dari hasil komputasi akhir ini.

Spesifikasi umum yang disyaratkan untuk fungsi dan algoritma kriptografi 3GPP adalah sebagai berikut:

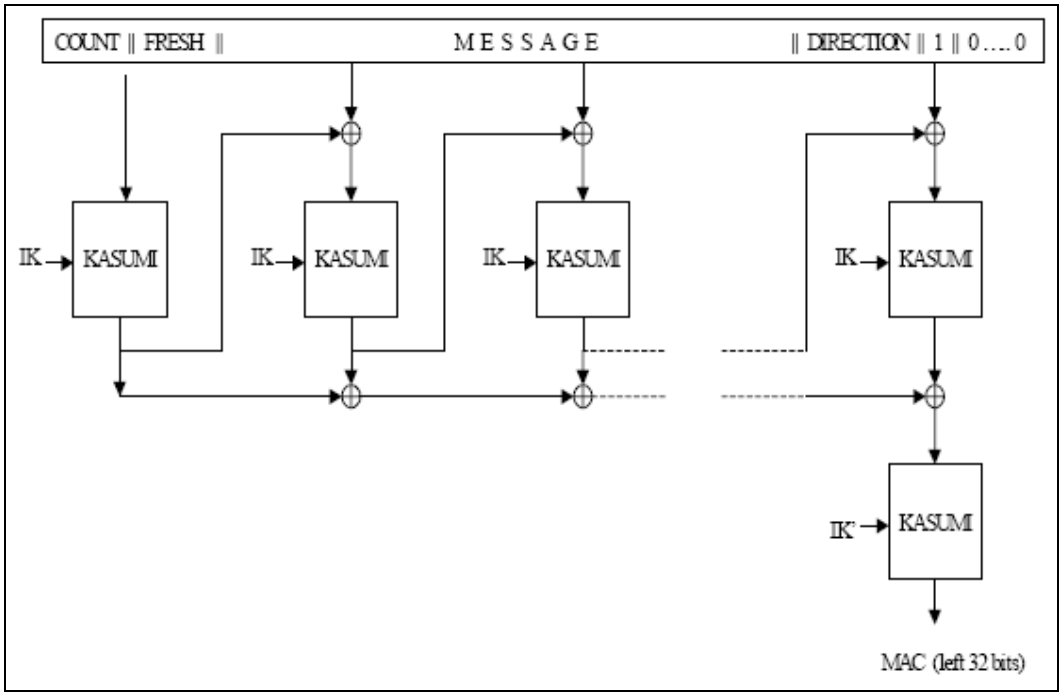
Generic requirements for 3GPP Cryptographic functions and algorithms

- Fungsi tersebut harus dirancang untuk pemakaian yang berkesinambungan paling tidak selama 20 tahun. Usaha untuk melakukan penyerangan dengan *workload* yang lebih kecil dari *exhaustive key search* melalui *effective key space* haruslah tidak mungkin untuk dilakukan.

- Perancang dari fungsi tersebut harus merancang algoritma yang mencerminkan kekuatan dari kebutuhan kualitatif di atas.

- Pelarangan pemakaian atau ekspor dari peralatan yang mengandung fungsi kriptografi dapat mencegah kemungkinan pemekaian peralatan tersebut pada negara tertentu.

- Diharapkan bahwa UE dan USIM yang mengandung algritma tersebut dapat bebas dari pelarangan penggunaan atau ekspor, dalam rangka untuk terjadinya peredaran bebas dari terminal 3G. Peralatan jaringan, termasuk RNC dan AuC mungkin akan berada pada batasan yang lebih ketat. Diharapkan bahwa RNC dan AuC yang mengandung algoritma tersebut dapat diekspor menurut kondisi yang terdapat pada Wasenaar Arrengement.



Gambar 2 Fungsi f9 pada Algoritma KASUMI

5. Operasi pada Algoritma KASUMI

KASUMI beroperasi pada 64-bit masukan **I** menggunakan kunci **K** 128-bit untuk menghasilkan keluaran 64-bit. Penjelasan selengkapnya akan dijelaskan di bawah ini.

Pertama, masukan I dibagi menjadi dua bagian L_0 dan R_0 masing-masing 32-bit, dimana

$$I = L_0 \,||\, R_0$$

Kemudian, untuk setiap integer i dengan $1 \le i \le 8$, didefinisikan:

$$R_i = L_{i-1},\ L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i)$$

Operasi ini akan menghasilkan keluaran $(L_8 \,||\, R_8)$ pada akhir iterasi kedelapan.

6. Komponen KASUMI

- Fungsi f_i

Fungsi ***fi( )*** mengambil masukan 32-bit ***I*** menghasilkan keluaran 32-bit ***O*** dikendalikan oleh round key ***RKi***, dimana round key tersebut terdiri dari subkey triplet dari (***KLi***, ***KOi***, ***KIi***). Fungsi ***fi( )*** itu sendiri terbentuk dari dua subfungsi; ***FL*** and ***FO*** yang berasosiasi dengan subkunci ***KLi*** (untuk ***FL***) dan subkey ***KOi*** dan ***KIi*** (untuk ***FO***).

Fungsi ***fi( )*** memiliki dua bentuk yang berbeda tergantung kepada iterasi yang sedang berjalan, apakah iterasi ganjil atau genap.
Untuk iterasi 1,3,5 and 7 didefinisikan:

$$fi(I,RKi) = FO(\ FL(\ I,\ KLi),\ KOi,\ KIi\ )$$

sedangkan untuk iterasi 2,4,6 and 8 didefinisikan:

$$fi(I,Ki) = FL(\ FO(\ I,\ KOi,\ KIi\ ),\ KLi\ )$$

Ilustrasi dari fungsi f_i ditunjukkan pada Gambar 3.

- Fungsi *FL*

Masukan dari fungsi *FL* terdiri dari 32-bit data masukan ***I*** dan 32bit subkey ***KLi***. Subkey ***KLi*** dibagi menjadi dua bagian, ***KLi,1*** and ***KLi,2*** dimana:

$$KLi = KLi,1 \,||\, KLi,2$$

Data masukan ***I*** juga dibagi dua, masing-masing 16-bit, ***L*** and ***R*** dimana:

$$I = L \,||\, R$$

Didefinisikan:

$$R' = R \oplus ROL\left(L \cap KL_{i,1}\right)$$

$$L' = L \oplus ROL\left(R' \cap KL_{i,2}\right)$$

Hasil keluaran dari fungsi ini yaitu sebesar 32-bit yang nilainya adalah:

$$I = L' \,||\, R'$$

Ilustrasi dari fungsi *FL* ditunjukkan pada Gambar 4.

- Fungsi *FO*

Masukan dari fungsi ini terdiri dari 32-bit data masukan I dan dua set dari subkunci, 48-bit subkunci KO_i and 48-bit subkunci KI_i.
Data masukan sebesar 32-bit tersebut dibagi menjadi dua bagian , L_0 and R_0, dengan:

$$I = L_0 \,||\, R_0$$

Subkunci sebesar 48-bit tersebut dibagi menjadi tiga subkunci sebesar 16-bit, dimana:

$$KO_i = KO_{i,1} \,||\, KO_{i,2} \,||\, KO_{i,3}$$
$$KI_i = KI_{i,1} \,||\, KI_{i,2} \,||\, KI_{i,3}$$

Kemudian untuk setiap integer j dengan $1 \le j \le 3$, didefinisikan:

$$R_j = FI\left(L_{j-1} \oplus KO_{i,j}, KI_{i,j}\right) \oplus R_{j-1}$$
$$L_j = R_{j-1}$$

Fungsi tersebut akan menghasilkan keluaran ($L3 \,||\, R3$) sebesar 32-bit.

Ilustrasi dari fungsi *FL* ditunjukkan pada Gambar 5.

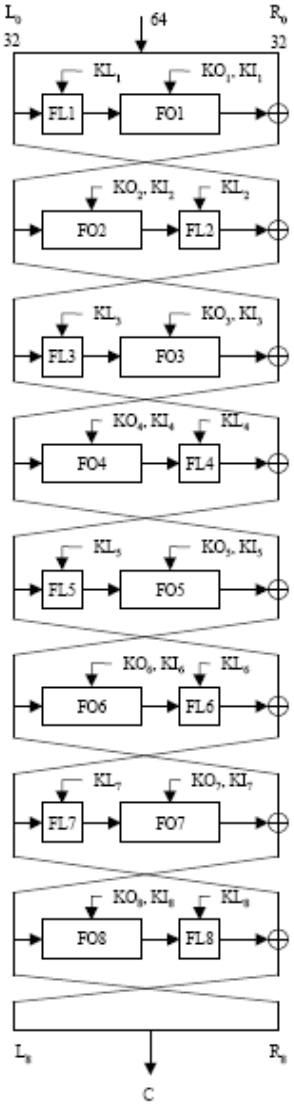
- Fungsi *FI*

Fungsi *FI* mengambil masukan 16-bit data masukan I dan 16-bit subkey $KI_{i,j}$. Masukan I dibagi menjadi dua bagian yang tidak sama panjang, 9-bit untuk bagian kiri L_0 and a 7-bit bagian kanan R_0 dimana:

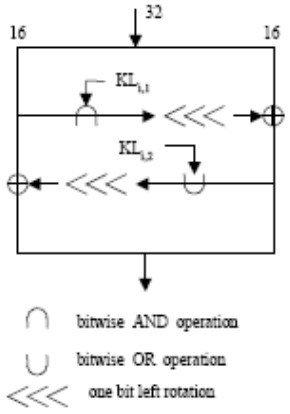
$$I = L_0 \,||\, R_0$$

Sama dengan data input, key $KI_{i,j}$ dibagi menjadi komponen 7-bit $KI_{i,j,1}$ dan komponen 9-bit $KI_{i,j,2}$ dimana:

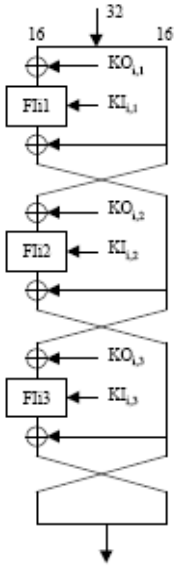
$$KI_{i,j} = KI_{i,j,1} \,||\, KI_{i,j,2}$$



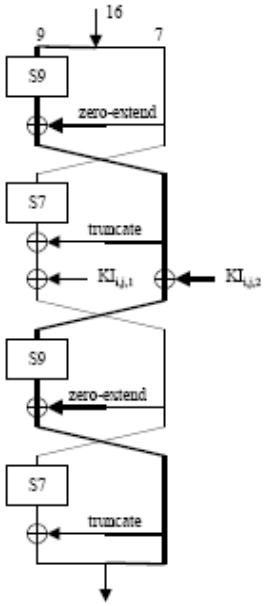
Gambar 3 KASUMI



Gambar 4 Fungsi FL



Gambar 5 Fungsi FO



Gambar 6 Fungsi FI

Fungsi *FI* menggunakan dua S-Box, ***S7*** yang memetakan masukan 7-bit menjadi keluaran 7-bit, dan ***S9*** yang memetakan masukan 9-bit menjadi keluaran 9-bit. Hal ini kan dijelaskan lebih rinci pada bagian selanjutnya. Fungsi ini juag menggunakan dua fungsi tambahan ***ZE( )*** dan ***TR( )***. Kedua fungsi tambahan tersebut didefinisikan sebagai berikut:

ZE(x): Mengambil masukan 7-bit dari ***x*** mengubahnya menjadi 9-bit dengan menambahkan dua 0 bit ke bagian MSB.
TR(x): Mengambil masukan 9-bit dari ***x*** mengubahnya menjadi 7-bit dengan menghilangkan dua 0 bit dari bagian MSB.

Selanjutnya didefinisikan operasi sebagai berikut ini:

$$L_1 = R_0$$
$$R_1 = S9[L_0] \oplus ZE(R_0)$$
$$L_2 = R_1 \oplus KI_{i,j,2}$$
$$R_2 = S7[L_1] \oplus TR(R_1) \oplus KI_{i,j,1}$$
$$L_3 = R_2$$
$$R_3 = S9[L_2] \oplus ZE(R_2)$$
$$L_4 = S7[L3] \oplus TR(R_3)$$
$$R_4 = R_3$$

Hasil dari fungsi ini yaitu (L_4 || R_4)sebesar 16-bit.

Ilustrasi dari fungsi *FL* ditunjukkan pada Gambar 6.

- S-box
KASUMI memiliki dua S-box yang telah dirancang untuk dapat diimplementasikan dengan mudah baik dalam logika kombinasi maupun dalam *look-up table*.
Masukan ***x*** terdiri dari tujuh atau sembilan bit dengan nomor bt yang sama pada keluaran ***y***.

x = x8 || x7 || x6 || x5 || x4 || x3 || x2 || x1 || x0
y = y8 || y7 || y6 || y5 || y4 || y3 || y2 || y1 || y0

dimana bit ***x8, y8*** dan ***x7, y7*** hanya digunakan pada **S9**, dan bit ***x0*** dan ***y0*** adalah LSB.

***S7*- S-Box**
Gerbang Logika:

```
y0 = x1x3 ⊕x4 ⊕x0x1x4 ⊕x5
⊕x2x5 ⊕x3x4x5 ⊕x6 ⊕x0x6
⊕x1x6 ⊕x3x6 ⊕x2x4x6 ⊕x1x5x6
⊕x4x5x6

y1 = x0x1 ⊕x0x4 ⊕x2x4 ⊕x5
⊕x1x2x5 ⊕x0x3x5 ⊕x6 ⊕x0x2x6
⊕x3x6 ⊕x4x5x6 ⊕1

y2 = x0 ⊕x0x3 ⊕x2x3⊕x1x2x4
⊕x0x3x4 ⊕x1x5 ⊕x0x2x5 ⊕x0x6
⊕x0x1x6 ⊕x2x6 ⊕x4x6⊕1

y3 = x1 ⊕x0x1x2 ⊕x1x4 ⊕x3x4
⊕x0x5 ⊕x0x1x5 ⊕x2x3x5
⊕x1x4x5 ⊕x2x6 ⊕x1x3x6

y4 = x0x2 ⊕x3 ⊕x1x3 ⊕x1x4
⊕x0x1x4 ⊕x2x3x4 ⊕x0x5
```

⊕x1x3x5 ⊕x0x4x5 ⊕x1x6 ⊕x3x6 ⊕x0x3x6 ⊕x5x6 ⊕1

y5 = x2 ⊕x0x2 ⊕x0x3 ⊕x1x2x3 ⊕x0x2x4 ⊕x0x5 ⊕x2x5 ⊕x4x5 ⊕x1x6 ⊕x1x2x6 ⊕x0x3x6 ⊕x3x4x6 ⊕x2x5x6 ⊕1

y6 = x1x2 ⊕x0x1x3⊕x0x4 ⊕x1x5 ⊕x3x5 ⊕x6 ⊕x0x1x6 ⊕x2x3x6 ⊕x1x4x6 ⊕x0x5x6

Tabel Desimal

54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18, 123, 33, 55, 113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25, 111, 124, 81, 53, 9, 121, 79, 52, 60, 58, 48, 101, 127, 40, 120, 104, 70, 71, 43, 20, 122, 72, 61, 23, 109, 13, 100, 77, 1, 16, 7, 82, 10, 105, 98, 117, 116, 76, 11, 89, 106, 0, 125, 118, 99, 86, 69, 30, 57, 126, 87, 112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35, 103, 32, 97, 28, 66, 102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29, 115, 44, 64, 107, 108, 24, 110, 83, 36, 78, 42, 19, 15, 41, 88, 119, 59, 3

Contoh:
Jika kita memiliki nilai masukan sebesar 38, maka dengan menggunakan tabel desimal S7[38], maka keluaran yang akan dihasilkan adalah 58.

Sedangkan untuk gerbang logika, kita jabarkan:
$38 = 0100110_2$, dimana x6 = 0, x5 = 1, x4 = 0, x3 = 0, x2 = 1, x1 = 1, x0 = 0, yang jika dimasukkan ke dalam rumus gerbang logika S7 akan menjadi sebagai berikut:

y0 = 0 ⊕ 0 ⊕0 ⊕1 ⊕1 ⊕0 ⊕0 ⊕ 0 ⊕ 0 ⊕0 ⊕0 ⊕0 ⊕0 = 0

y1 = 0 ⊕ 0 ⊕ 0 ⊕1 ⊕1 ⊕ 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 1

y2 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 0

y3 = 1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 = 1

y4 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 1

y5 = 1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 1

y6 = 1 ⊕0 ⊕0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 = 0

Maka, $y = 0111010_2 = 58$.

***S9*- S-Box**

y0 = x0x2 ⊕x3 ⊕x2x5 ⊕x5x6 ⊕x0x7 ⊕x1x7 ⊕x2x7 ⊕x4x8 ⊕x5x8 ⊕x7x8 ⊕1

y1 = x1 ⊕x0x1 ⊕x2x3 ⊕x0x4 ⊕x1x4 ⊕x0x5 ⊕x3x5 ⊕x6 ⊕x1x7 ⊕x2x7 ⊕x5x8 ⊕1

y2 = x1 ⊕x0x3 ⊕x3x4 ⊕x0x5 ⊕x2x6 ⊕x3x6 ⊕x5x6 ⊕x4x7 ⊕x5x7 ⊕x6x7 ⊕x8 ⊕x0x8 ⊕1

y3 = x0 ⊕x1x2 ⊕x0x3 ⊕x2x4 ⊕x5 ⊕x0x6 ⊕x1x6 ⊕x4x7 ⊕x0x8 ⊕x1x8 ⊕x7x8

y4 = x0x1 ⊕x1x3 ⊕x4 ⊕x0x5 ⊕x3x6 ⊕x0x7 ⊕x6x7 ⊕x1x8 ⊕x2x8 ⊕x3x8

y5 = x2 ⊕x1x4 ⊕x4x5 ⊕x0x6 ⊕x1x6 ⊕x3x7 ⊕x4x7 ⊕x6x7 ⊕x5x8 ⊕x6x8 ⊕x7x8 ⊕1

y6 = x0 ⊕x2x3 ⊕x1x5 ⊕x2x5 ⊕x4x5 ⊕x3x6 ⊕x4x6 ⊕x5x6 ⊕x7 ⊕x1x8 ⊕x3x8 ⊕x5x8 ⊕x7x8

y7 = x0x1 ⊕x0x2 ⊕x1x2 ⊕x3 ⊕x0x3 ⊕x2x3 ⊕x4x5 ⊕x2x6 ⊕x3x6 ⊕x2x7 ⊕x5x7 ⊕x8 ⊕1

y8 = x0x1 ⊕x2 ⊕x1x2 ⊕x3x4 ⊕x1x5 ⊕x2x5 ⊕x1x6 ⊕x4x6 ⊕x7 ⊕x2x8 ⊕x3x8

Tabel Desimal:

167, 239, 161, 379, 391, 334, 9, 338, 38, 226, 48, 358, 452, 385, 90, 397, 183, 253, 147, 331, 415, 340, 51, 362, 306, 500, 262, 82, 216, 159, 356, 177, 175, 241, 489, 37, 206, 17, 0, 333, 44, 254, 378, 58, 143, 220, 81, 400, 95, 3, 315, 245, 54, 235, 218, 405, 472, 264, 172, 494, 371, 290, 399, 76, 165, 197,395, 121, 257, 480, 423, 212, 240, 28, 462, 176, 406, 507, 288, 223, 501, 407, 249, 265, 89, 186, 221, 428, 164, 74, 440, 196, 458, 421, 350, 163, 232, 158, 134, 354, 13, 250, 491, 142, 191, 69, 193, 425, 152, 227, 366, 135, 344, 300, 276, 242, 437, 320, 113,

278, 11, 243, 87, 317, 36, 93,
496, 27, 487, 446, 482, 41, 68,
156, 457, 131, 326, 403, 339,
20, 39, 115, 442, 124, 475, 384,
508, 53, 112, 170, 479, 151,
126, 169, 73, 268, 279, 321,
168, 364, 363, 292, 46, 499,
393, 327, 324, 24, 456, 267,
157, 460, 488, 426, 309, 229,
439, 506, 208, 271, 349, 401,
434, 236, 16, 209, 359, 52, 56,
120, 199, 277, 465, 416, 252,
287, 246, 6, 83, 305, 420, 345,
153, 502, 65, 61, 244, 282, 173,
222, 418, 67, 386, 368, 261,
101, 476, 291, 195, 430, 49, 79,
166, 330, 280, 383, 373, 128,
382, 408, 155, 495, 367, 388,
274, 107, 459, 417, 62,454, 132,
225, 203, 316, 234, 14, 301, 91,
503, 286, 424, 211, 347, 307,
140,374, 35, 103, 125, 427, 19,
214, 453, 146, 498, 314, 444,
230,256, 329, 198, 285, 50, 116,
78, 410, 10, 205, 510, 171, 231,
45, 139, 467, 29, 86, 505, 32,
72, 26, 342, 150, 313, 490, 431,
238, 411, 325, 149, 473, 40,
119, 174, 355, 185, 233, 389,
71, 448, 273, 372, 55, 110, 178,
322, 12, 469, 392, 369, 190, 1,
109, 375, 137, 181, 88, 75, 308,
260, 484, 98, 272, 370, 275,
412, 111, 336, 318, 4, 504, 492,
259, 304, 77, 337, 435, 21, 357,
303, 332, 483, 18, 47, 85, 25,
497, 474, 289, 100, 269, 296,
478, 270, 106, 31, 104, 433, 84,
414, 486, 394, 96, 99, 154, 511,
148, 413, 361, 409, 255, 162,
215, 302, 201, 266, 351, 343,
144, 441, 365, 108, 298, 251,
34, 182, 509, 138, 210, 335,
133, 311, 352, 328, 141, 396,
346, 123, 319, 450, 281, 429,
228, 443, 481, 92, 404, 485,
422, 248, 297, 23, 213, 130,
466, 22, 217, 283, 70, 294, 360,
419, 127, 312, 377, 7, 468, 194,
2, 117, 295, 463, 258, 224, 447,
247, 187, 80,398, 284, 353, 105,
390, 299, 471, 470, 184, 57,
200, 348, 63, 204, 188, 33, 451,
97, 30, 310, 219, 94, 160, 129,
493, 64, 179, 263, 102, 189,
207, 114, 402, 438, 477, 387,
122, 192, 42, 381, 5, 145, 118,
180, 449, 293, 323, 136, 380,
43, 66, 60, 455, 341, 445, 202,
432, 8, 237, 15, 376, 436, 464,
59, 461

Contoh:
Jika kita memiliki nilai masukan sebesar 138, maka dengan menggunakan tabel desimal S9[138], maka keluaran yang akan dihasilkan adalah 339.

Sedangkan untuk gerbang logika, kita jabarkan:
$138 = 010001010_2$, dimana x8 = 0, x7 = 1, x6 = 0, x5 = 0, x4 = 0, x3 = 1, x2 = 0, x1 = 1, x0 = 0, yang jika dimasukkan ke dalam rumus gerbang logika S9 akan menjadi sebagai berikut:

y0 = 0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕1 ⊕0
⊕0 ⊕0 ⊕0 ⊕1 = 1
y1 = 1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0
⊕0 ⊕1 ⊕0 ⊕0 ⊕1 = 1
y2 = 1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0
⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 0
y3 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0
⊕0 ⊕0 ⊕0 = 0
y4 = 0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0
⊕0 ⊕0 ⊕0 = 1
y5 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 ⊕0 ⊕0
⊕0 ⊕0 ⊕0 ⊕1 = 0
y6 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕ 0 ⊕0
⊕0 ⊕1 ⊕0 ⊕0 ⊕0 ⊕0 = 1
y7 = 0 ⊕0 ⊕0 ⊕1 ⊕ 0 ⊕0 ⊕0
⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕1 = 0
y8 = 0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0 ⊕0
⊕0 ⊕1 ⊕0 ⊕0 = 1

Maka, $y = 101010011_2 = 339$.

7. Penjadwalan Kunci
KASUMI memiliki kunci ***K*** sebesar 128-bit. Untuk setiap iterasi, KASUMI menggunakan kunci 128-bit yang diturunkan dari ***K***.

Kunci K pertama dibagi menjadi 16-bit sebanyak 8 buah K_1, .., K_8 dimana:

$$\boldsymbol{K = K_1 \,||\, K_2 \,||\, K_3 \,||\ldots||\, K_8}$$

Array dari subkey kedua, $\boldsymbol{K_j'}$ diturunkan dari $\boldsymbol{K_j}$ dengan cara:
Untuk setiap integer $\boldsymbol{j}$ dengan $1 \le \boldsymbol{j} \le 8$, maka:

$$\boldsymbol{K_j' = K_j \oplus C_j}$$

Dimana $\boldsymbol{C_j}$ adalah nilai konstan yang terdapat pada Tabel 1.

Round subkeys diturunkan dari $\boldsymbol{K_j}$ and $\boldsymbol{K_j'}$ sesuai dengan aturan pada Tabel 2.

8. Evaluasi Algoritma KASUMI

Algoritma KASUMI dievaluasi oleh tiga kelompok evaluasi yang bersifat independen. Evaluasi yang dilakukan meliputi analisis terhadap algoritma KASUMI sebagai

algoritma block cipher 64-bit beserta fungsi integritas dan enkripsi (f8 dan f9). Hasil evaluasi lengkap dapat dibaca pada [6]. Makalah ini hanya akan mengangkat ringkasan dari kesimpulan yang dihasilkan oleh ketiga kelompok evaluasi tersebut.

Tabel 1 Nilai Konstanta C_i

C1	0x0123
C2	0x4567
C3	0x89AB
C4	0xCDEF
C5	0xFEDC
C6	0xBA98
C7	0x7654
C8	0x3210

Tabel 2 Tabel Round Subkey

	1	2	3	4	5	6	7	8
$KL_{i,1}$	K1<<<1	K2<<<1	K3<<<1	K4<<<1	K5<<<1	K6<<<1	K7<<<1	K8<<<1
$KL_{i,2}$	K3′	K4′	K5′	K6′	K7′	K8′	K1′	K2′
$KO_{i,1}$	K2<<<5	K3<<<5	K4<<<5	K5<<<5	K6<<<5	K7<<<5	K8<<<5	K1<<<5
$KO_{i,2}$	K6<<<8	K7<<<8	K8<<<8	K1<<<8	K2<<<8	K3<<<8	K4<<<8	K5<<<8
$KO_{i,3}$	K7<<<13	K8<<<13	K1<<<13	K2<<<13	K3<<<13	K4<<<13	K5<<<13	K6<<<13
$KI_{i,1}$	K5′	K6′	K7′	K8′	K1′	K2′	K3′	K4′
$KI_{i,2}$	K4′	K5′	K6′	K7′	K8′	K1′	K2′	K3′
$KI_{i,3}$	K8′	K1′	K2′	K3′	K4′	K5′	K6′	K7′

- Evaluator-1

Algoritma f8

Perancangan dari f8 dianggap cukup baik, dengan jenis serangan terbaik hanya yang berbasiskan collision dan memerlukan pengetahuan blok plainteks sebanyak 2^{32}. Tetapi fungsi ini tidak direkomendasikan untuk digunakan selain untuk yang sudah dispesifikasikan oleh 3GPP.

Algoritma f9

Fungsi f9 dinilai cukup baik dan telah diuji dengan menggunakan teori dekorelasi. Penyerangan terhadap fungsi ini dipandang dari perspektif kriptanalisis memerlukan 2^{64} *chosen messages*, sehingga dianggap cukup aman. Mereka juga menganggap bahwa jika ada jenis serangan yang lebih baik, tetap tidak akan berbahaya bagi rancangan yang dibuat oleh 3GPP.

Block Cipher

Mereka dapat melakukan serangan terhadap kunci yang digunakan untuk versi Algoritma

KASUMI yang hanya menggunakan 6 iterasi menggunakan *"academic" attack*, dan setelah mempelajari penjadwalan kunci dapat dilakukan dalam 5 iterasi. Tetapi mereka tidak dapat melakukan serangan tersebut terhadap versi sempurna dari algoritma.

- Evaluator-2

Algoritma f8 dan f9

Mereka memiliki keraguan mengenai algoritma f8 dan f9 yang digunakan karena memiliki beberapa hal yang janggal dan tidak biasa, tetapi tidak melakukan penyelidikan ataupun pengujian statistik dengan lengkap.

Block Cipher

Mereka telah melakukan beberapa jenis serangan, seperti *Divide-and-Conquer, Meet-in-the-Middle, Differential Cryptanalysis beserta variannya, Linear Cryptanalysis, Related Key, serta Weak Key Classes*, dan menyimpulkan bahwa mereka tidak menemukan kelemahan yang berarti dari algoritma ini. Tetapi mereka meragukan keputusan untuk mengubah rancangan MISTY1 untuk membuat algoritma KASUMI ini.

- Evaluator-3

Block Cipher

Mereka tidak menemukan cacat atau kelemahan pada keamanan dari KASUMI. Ukuran kunci yang digunakan sebesar 128-bit dinilai cukup aman dari serangan *exhaustive search* selama 20 tahun ke depan.

Algoritma f8 dan f9

Mereka tidak menemukan kelemahan pada mode enkripsi dan integritas karena dinilai telah memiliki tingkat keamanan yang cukup tinggi.

Kesimpulan Hasil Evaluasi

Kesimpulan secara umum yaitu bahwa algoritma KASUMI dinilai dirancang berdasarkan prinsip yang baik, dan tidak ditemukan jenis serangan yang cukup praktis untuk menembusnya. Dan algoritma KASUMI dinilai sesuai untuk penggunaan yang dimaksudkan untuknya.

9. Kode Simulasi Algoritma KASUMI

- Fungsi FI

Kode yang digunakan untuk mensimulasikan fungsi dari FI dapat dilihat pada Kode 1. Pada kode tersebut, fungsi FI menerima masukan data dan subkunci yang masing-masing sebesar sebesar 16-bit. Data masukan yang diterimanya itu kemudian dibagi menjadi dua bagian. Bagian pertama yaitu 9-bit pertama dari masukan, sedangkan bagian kedua yaitu 7-bit berikutnya. Selanjutnya dilakukan berbagai operasi yang telas dibahas pada bagian sebelumnya. Terakhir, fungsi ini menghasilkan keluaran sebesar 16-bit yang akan digunakan oleh fungsi FO selanjutnya.

- Fungsi FL

Kode yang digunakan untuk mensimulasikan fungsi dari FL dapat dilihat pada Kode 1. Pada kode tersebut, fungsi FI menerima masukan berupa 32-bit data beeserta index yang digunakan untuk mengetahui subkunci mana yang akan dipakai. Pertama, data masukan akan dibagi menjadi dua bagian yang besarnya masing-masing 16-bit. Selanjutnya akan dilakukan berbagai operasi yang sudah dibahas pada bagian sebelumnya. Fungsi ini akan

menghasilkan 32-bit keluaran yang akan dipakai sebagai masukan untuk fungsi FO.

- Fungsi FO

Kode yang digunakan untuk mensimulasikan fungsi Fo dapat dilihat pada Kode 3. Pada kode simulasi tersebut, sama seperti fungsi FL, fungsi ini juga menerima masukan data sebesar 32-bit beserta indeks yang digunakan untuk mengetahui subkunci yang akan digunakan pada iterasi yang sedang berlangsung. Fungsi ini juga membagi data masukannya menjadi dua buah bagian sebesar 16-bit yang akan menjadi masukan dari fungsi FI secara bergantian kiri dan kanan.

- KASUMI (Main Method)

Kode yang digunakan untuk mensimulasikan metode utama dari program simulasi KASUMI dapat dilihat pada Kode 4. Pada fungsi utama ini, pertama kali menerima masukan sebesar 64-bit, yang kemudian akan dipecah menjadi dua bagian yang besarnya masing-masing 32-bit. Selanjutnya akan dilakukan operasi jaringan feistel yang berisi fungsi FO dan FL. Operasi jaringan feistel ini dilakukan sebanyak 8 kali iterasi. Setelah semua fungsi yang terdapat di dalamnya selesai dijalankan, maka fungsi ini akan menghasilkan keluaran data akhir yang diinginkan.

- Key Schedule

Kode yang digunakan untuk mensimulasikan metode utama dari program simulasi KASUMI dapat dilihat pada Kode 5. Fungsi ini membentuk penjadwalan kunci yang akan digunakan dalam program simulasi KASUMI ini. Fungsi ini menerima masukan kunci yang digunakan, untuk diproses menghasilkan 8 buah subkunci penggunaannya akan dijadwalkan sesuai dengan iterasi yang sedang berlangsung.

```
/*---------------------------------------------------------------------
* FI()
* The FI function (fig 3). It includes the S7 and S9 tables.
* Transforms a 16-bit value.
*---------------------------------------------------------------------*/

static ul6 FI( ul6 in, ul6 subkey )
{
    ul6 nine, seven;
    static ul6 S7[] = {
        54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18,123, 33,
        55,113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25,111,124, 81,
        53, 9,121, 79, 52, 60, 58, 48,101,127, 40,120,104, 70, 71, 43,
        20,122, 72, 61, 23,109, 13,100, 77, 1, 16, 7, 82, 10,105, 98,
        117,116, 76, 11, 89,106, 0,125,118, 99, 86, 69, 30, 57,126, 87,
        112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35,103, 32, 97, 28, 66,
        102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29,115, 44,
        64,107,108, 24,110, 83, 36, 78, 42, 19, 15, 41, 88,119, 59, 3};
    static ul6 S9[] = {

    /* The sixteen bit input is split into two unequal halves, *
    * nine bits and seven bits - as is the subkey */

    nine = (ul6)(in>>7);
    seven = (ul6)(in&0x7F);

    /* Now run the various operations */

    nine = (ul6)(S9[nine] ^ seven);
    seven = (ul6)(S7[seven] ^ (nine & 0x7F));

    seven ^= (subkey>>9);
    nine ^= (subkey&0x1FF);

    nine = (ul6)(S9[nine] ^ seven);
    seven = (ul6)(S7[seven] ^ (nine & 0x7F));

    in = (ul6)((seven<<9) + nine);

    return( in );
}
```

Kode 1 - Kode Simulasi Fungsi FI()

```
/*---------------------------------------------------------------------
* FL()
* The FL() function.
* Transforms a 32-bit value. Uses <index> to identify the
* appropriate subkeys to use.
*---------------------------------------------------------------------*/

static u32 FL( u32 in, int index )
{
    ul6 l, r, a, b;

    /* split out the left and right halves */

    l = (ul6)(in>>16);
    r = (ul6)(in);

    /* do the FL() operations */

    a = (ul6) (l & KLil[index]);
    r ^= ROL16(a,1);

    b = (ul6)(r | KLi2[index]);
    l ^= ROL16(b,1);

    /* put the two halves back together */

    in = (l<<16) + r;

    return( in );
}
```

Kode 2 - Kode Simulasi Fungsi FL()

```
/*------------------------------------------------------------------
* FO()
* The FO() function.
* Transforms a 32-bit value. Uses <index> to identify the
* appropriate subkeys to use.
*------------------------------------------------------------------*/

static u32 FO( u32 in, int index )
{
    u16 left, right;

    /* Split the input into two 16-bit words */

    left = (u16)(in>>16);
    right = (u16) in;

    /* Now apply the same basic transformation three times */

    left ^= KOi1[index];
    left = FI( left, KIi1[index] );
    left ^= right;

    right ^= KOi2[index];
    right = FI( right, KIi2[index] );
    right ^= left;

    left ^= KOi3[index];
    left = FI( left, KIi3[index] );
    left ^= right;

    in = (right<<16)+left;

    return( in );
}
```

Kode 3 – Kode Simulasi Fungsi FO()

```
/*------------------------------------------------------------------
* Kasumi()
* the Main algorithm (fig 1). Apply the same pair of operations
* four times. Transforms the 64-bit input.
*------------------------------------------------------------------*/

void Kasumi( u8 *data )
{
    u32 left, right, temp;
    DWORD *d;
    int n;
    /* Start by getting the data into two 32-bit words (endian corect) */

    d = (DWORD*)data;
    left = (d[0].b8[0]<<24)+(d[0].b8[1]<<16)+(d[0].b8[2]<<8)+(d[0].b8[3]);
    right = (d[1].b8[0]<<24)+(d[1].b8[1]<<16)+(d[1].b8[2]<<8)+(d[1].b8[3]);

    n = 0;

    do{ temp = FL( left, n );
        temp = FO( temp, n++ );
        right ^= temp;
        temp = FO( right, n );
        temp = FL( temp, n++ );
        left ^= temp;
    } while ( n<=7 );

    /* return the correct endian result */

    d[0].b8[0] = (u8)(left>>24); d[1].b8[0] = (u8)(right>>24);
    d[0].b8[1] = (u8)(left>>16); d[1].b8[1] = (u8)(right>>16);
    d[0].b8[2] = (u8)(left>>8); d[1].b8[2] = (u8)(right>>8);
    d[0].b8[3] = (u8)(left); d[1].b8[3] = (u8)(right);
}

/*
```

Kode 4 – Kode Simulasi KASUMI

```
/*---------------------------------------------------------------------
* KeySchedule()
* Build the key schedule. Most "key" operations use 16-bit
* subkeys so we build ul6-sized arrays that are "endian" correct.
*---------------------------------------------------------------------*/

void KeySchedule( u8 *k )
{
    static ul6 C[] = {
    0x0123,0x4567,0x89AB,0xCDEF, 0xFEDC,0xBA98,0x7654,0x3210 };
    ul6 key[8], Kprime[8];
    WORD *kl6;
    int n;

    /* Start by ensuring the subkeys are endian correct on a 16-bit basis */

    kl6 = (WORD *)k;
    for( n=0; n<8; ++n )
        key[n] = (ul6)((kl6[n].b8[0]<<8) + (kl6[n].b8[1]));

    /* Now build the K'[] keys */

    for( n=0; n<8; ++n )
        Kprime[n] = (ul6)(key[n] ^ C[n]);

    /* Finally construct the various sub keys */

    for( n=0; n<8; ++n )
    {
        KLil[n] = ROL16(key[n],1);
        KLi2[n] = Kprime[(n+2)&0x7];
        KOil[n] = ROL16(key[(n+1)&0x7],5);
        KOi2[n] = ROL16(key[(n+5)&0x7],8);
        KOi3[n] = ROL16(key[(n+6)&0x7],13);
        KIil[n] = Kprime[(n+4)&0x7];
        KIi2[n] = Kprime[(n+3)&0x7];
        KIi3[n] = Kprime[(n+7)&0x7];
    }
}
```

Kode 5 – Kode Simulasi Penjadwalan Kunci

Daftar Pustaka

[1] 3GPP Task Force, *Specification of the 3GPP Confidentialty and Integrity Algorithms. Document 2: KASUMI Algorithm Specification,* 1999.

[2] David Cereso's Weblog, "On GSM Security", URL: http://www.cerezo.name/weblog/

[3] GSM Wrold, "GSM calls even more secure thanks to new A5/3 Algorithm", URL: http://www.gsmworld.com/news/press_2002/press_15.shtml

[4] Matsui, Mitsuru, Toshio Tokita, *MISTY, KASUMI and Camellia CipherAlgorithm Development*, 2002.

[5] Preneel, Bart, "Mobile Network Security", 2004.

[6] Security Algorithms Group of Experts (SAGE), *Report on the Evaluation of 3GPP Standard Confidentialty and Integrity Algorithms*, 2000.