STUDI DAN ANALISIS *GRAIN CIPHER*

Stephen Herlambang – NIM : 13507040

Program Studi Teknik Informatika - Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
e-mail: stephen_herlambang@yahoo.co.id

Abstrak

Ketika mendesain algoritma kriptografi, banyak hal yang harus dipertimbangkan. Hal-hal tersebut adalah kecepatan, keamanan, dan kesederhanaan. Pada beberapa kasus, diperlukan algoritma kriptografi yang memiliki kompleksitas hardware yang rendah. Hal ini misalnya pada kasus *RFID tag* dimana memori dan daya sangat terbatas. Algoritma seperti AES tidak layak untuk diimplementasikan pada *RFID tag* karena memiliki kompleksitas yang tinggi.
Grain cipher merupakan algoritma stream cipher yang didesain untuk dapat diimplementasikan pada perangkat keras yang memiliki keterbatasan (misalnya memori dan daya). Algoritma ini menerima kunci 80-bit dan IV *(initial value)* 64-bit. Grain cipher dibuat oleh Martin Hell, Thomas Johansson, dan Willi Meier. Algoritma ini diikusertakan dan dipilih untuk final proyek eSTREAM.
Makalah ini akan membahas studi mengenai Grain Cipher, analisisnya termasuk serangan-serangan pada Grain Cipher, dan perbandingannya dengan *hardware-oriented* cipher lainnya seperti A5/1 dan Trivium.

Kata kunci: *Grain cipher, LFSR, NFSR, A5/1, Trivium.*

1. Pendahuluan

Pada zaman sekarang ini, kebutuhan akan informasi semakin berkembang. Dan dalam pengiriman maupun penerimaan informasi itu, dibutuhkan keamanan. Salah satu cara agar berita yang dikirim aman adalah dengan menggunakan enkripsi agar bila pesan dapat disadap, pihak musuh tidak dapat dengan mudah mengetahui pesan yang dikirim apalagi pesan yang nilai kerahasiaannya sangat besar.

Ada dua macam sistem *cipher* menurut kunci yang dipakai yaitu *cipher* simetrik dan *cipher* asimetrik. Sistem *cipher* simetrik masih dibagi menjadi dua lagi yaitu *block cipher* dan stream *cipher*. *Stream cipher* digunakan karena relatif cepat dibanding dengan *block cipher*.

Stream cipher adalah *cipher* simetrik yang beroperasi dengan transformasi waktu yang bervariasi pada plain teks sacara individu. *Stream cipher* yang paling banyak dikenal adalah LFSR dan NLFSR. Perbedaan dari keduanya adalah fungsi yang dipakai linear dan non linear. LFSR mudah dianalisis dengan kriptanalisis karena sifat dari LFSR yang cenderung linear. Cara untuk menangani hal itu yaitu dengan mengkombinasikannya menjadi fungsi yang tidak linear agar tahan terhadap serangan yang bersifat linear.

Terdapat beberapa faktor yang menentukan apakah suatu algoritma kriptografi baik atau tidak. Hal-hal tersebut misalnya kecepatan, keamanan dan kesederhanaan. Membandingkan beberapa *cipher*, terdapat kemungkinan bahwa yang *cipher* yang satu lebih cepat pada prosesor 32-bit, sedangkan yang lain lebih cepat pada perangkat keras dengan prosesor 8-bit. Kesederhanaan dari desain adalah faktor lain yang harus diperhitungkan.

Terdapat kebutuhan untuk *cipher* yang memiliki kompleksitas perangkat keras yang sangat rendah. Suatu *RFID (Radio Frequency Identification) tag* adalah sebuah contoh suatu produk di mana jumlah memori dan daya sangat terbatas. Penggunaan *RFID tag* tersebut dapat memiliki konsekuensi yang cukup besar apabila telah berkaitan dengan transaksi-transaksi seperti pembayaran elektronik. Oleh karena itu, diperlukan algoritma kriptografi yang diimplementasikan pada *tag* tersebut. Implementasi *cipher* seperti AES pada *RFID tag* tidak layak karena *cipher* ini memiliki kompleksitas yang tinggi, yaitu banyak gerbang (*gate*) diperlukan.

2. *Grain Cipher*

Banyak stream cipher yang didasarkan pada *linear feedback shift registers (LFSR)*, tidak hanya untuk sifat statistik yang baik dari urutan yang mereka produksi, tetapi juga untuk kesederhanaan dan kecepatan dari implementasi ke perangkat keras. Beberapa stream cipher terbaru yang berbasis LSFR berorientasi pada *word*. Hal ini menyebabkan implementasi pada perangkat lunak lebih efisien tetapi pada perangkat keras *cipher* yang berorientasi pada *word* memiliki kompleksitas yang lebih tinggi daripada *cipher* yang berorientasi pada *bit*. Grain

cipher didesain berdasarkan *bit-oriented shift register* dengan tambahan yang memungkinkan peningkatan kecepatan berbanding lurus dengan kemampuan perangkat keras yang tersedia.

Grain cipher adalah *bit oriented synchronous stream cipher*. Pada *synchronous stream cipher, keystream* dibangkitkan secara terpisah dari *plaintext*. Grain cipher didasarkan pada dua *shift register*, satu dengan *linear feedback (LFSR)* dan satu dengan *nonlinear feedback (NFSR)*. LSFR menjamin perioda minimum untuk *keystream* dan juga menyediakan keseimbangan pada keluaran. NFSR bersama-sama dengan *filter* nonlinear menyebabkan ketidaklinearitasan pada cipher. Masukan pada NFSR ditutupi (*masked*) dengan keluaran dari LFSR sehingga kondisi dari NFSR menjadi seimbang. Kedua *shift register* masing-masing memiliki panjang 80 bit dan panjang IV (*Initial Value*) adalah 64 bit.

2.1 Cara Kerja *Grain Cipher*

Grain cipher terdiri dari tiga bagian utama, yaitu sebuah LFSR, sebuah NFSR, dan sebuah fungsi penyaring (*filter)*.

Linear feedback shift register (LFSR) adalah shift register yang bit masukannya merupakan fungsi linear dari state sebelumnya. Satu-satunya fungsi linear pada bit satuan adalah xor, oleh karena itu LFSR adalah shift register yang bit masukannya dibangkitkan oleh *exclusive-or* (xor) dari beberapa bit dari keseluruhan nilai shift register.

Initial value dari LFSR dikenal dengan seed, dan karena operasi dari *register* bersifat deterministik, aliran nilai yang dihasilkan oleh *register* akan sepenuhnya ditentukan oleh state sekarang atau sebelumnya. Dengan begitu, karena *register* memiliki jumlah state yang terbatas, pasti akan terbentuk siklus yang berulang. Akan tetapi, LFSR yang memiliki fungsi umpan balik yang baik dapat memproduksi sekuens bit yang tampak acak dan memiliki siklus yang sangat panjang.

Pada grain cipher, isi dari LSFR dapat dinotasikan dengan:

$$s_i, s_{i+1}, s_{i+2}, \ldots, s_{i+78}, s_{i+79}$$

Fungsi umpan balik dari LFSR, f(x), adalah fungsi polinom berderajat 80. Fungsi ini dapat didefinisikan sebagai berikut:

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$$

Fungsi *update* dari LSFR adalah sebagai berikut:

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i$$

dan isi dari NFSR dapat dinotasikan dengan;

$$b_i, b_{i+1}, b_{i+2}, \ldots, b_{i+78}, b_{i+79}$$

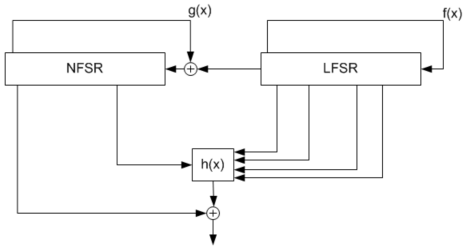
Fungsi umpan balik dari NFSR, g(x), didefinisikan sebagai berikut:

$$\begin{aligned} g(x) = 1 &+ x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} \\ &+ x^{52} + x^{59} + x^{65} + x^{71} + x^{80} \\ &+ x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} \\ &+ x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} \\ &+ x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} \\ &+ x^{17}x^{20}x^{59}x^{65} \\ &+ x^{17}x^{20}x^{28}x^{35}x^{43} \\ &+ x^{47}x^{52}x^{59}x^{65}x^{71} \\ &+ x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \end{aligned}$$

Fungsi *update* dari NFSR dapat didefinisikan sebagai berikut:

$$\begin{aligned} &b_{i+80} \\ &= s_i + b_{i+63} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} \\ &+ b_{i+33} + b_{i+28} + b_{i+21} + b_{i+15} + b_{i+9} + b_i \\ &+ b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} \\ &+ b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} \\ &+ b_{i+63}b_{i+45}b_{i+28}b_{i+9} + b_{i+60}b_{i+52}b_{i+37}b_{i+33} \\ &+ b_{i+63}b_{i+60}b_{i+21}b_{i+15} \\ &+ b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} \\ &+ b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} \\ &+ b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21} \end{aligned}$$

Cara kerja dari *grain cipher* secara umum dapat digambarkan dengan diagram sebagai berikut:



Gambar 1 Diagram Umum *Grain Cipher*

Isi dari kedua *shift register* merepresentasikan kondisi / state dari cipher. Dari kondisi ini, lima variabel diambil sebagai masukan ke fungsi *boolean* h(x). Fungsi filter ini dipilih untuk seimbang, bebas korelasi dengan urutan pertama dan memiliki derajat aljabar 3. Ketidaklinearitasan adalah paling memungkinkan untuk fungsi ini.

Masukan diambil baik dari LFSR dan dari NFSR. Fungsi tersebut dapat didefinisikan sebagai berikut:

$$\begin{aligned} h(x) = x_1 &+ x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 \\ &+ x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 \\ &+ x_2x_3x_4 \end{aligned}$$

dimana pengubah x_0, x_1, x_2, x_3 dan x_4 berkorespondensi dengan s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64} dan b_{i+63}.

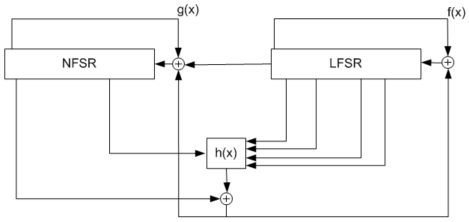
Keluaran dari fungsi filter ini di-xor (*masked*) dengan bit b_i dari NFSR untuk mendapatkan keystream.

2.2 Inisialisasi Kunci

Sebelum tiap *keystream* dibangkitkan, *cipher* harus diinisialisasi dengan kunci dan IV (*Initial*

Value). Misalkan bit-bit dari kunci, k, dilambangkan dengan k_i, dengan 0 <= i <= 79, dan bit-bit dari IV dilambangkan dengan IV_i, dengan 0 <= i <= 63.
Inisialisasi kunci dlakukan dengan cara sebagai berikut:

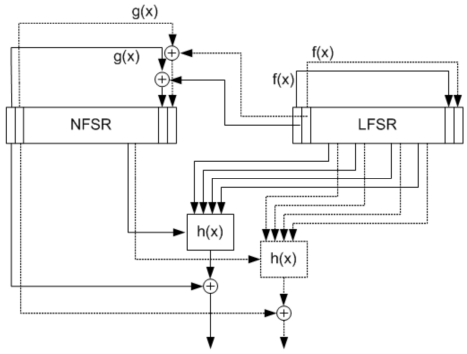
1. NFSR diisi dengan bit-bit kunci, $b_i = k_i$, 0 <= i <= 79.
2. 64 bit pertama dari LFSR diisi dengan IV, $s_i = IV_i$, 0 <= i <= 63.
3. Sisa bit dari LFSR diisi dengan satu, $s_i = 1$, 64 <= i <= 79.
   Karena itu, LFSR tidak dapat diinisialisasi dengan seluruhnya kondisi 0 (*zero state*).
4. Cipher di-*clock* 160 kali tanpa membuat satupun *running key*. Keluaran dai fungsi filter dikembalikan dan di-xor dengan masukan, baik ke LFSR dan ke NFSR, seperti pada gambar di bawah ini.



Gambar 2 Inisialisasi *Grain Cipher*

2.3 *Throughput*

Kedua shift register di-clock secara teratur sehingga cipher akan menghasilkan *output* / keluaran 1 bit/*clock*. Dimungkinkan untuk meningkatkan kecepatan dari cipher dengan meningkatkan kemampuan perangkat keras. Hal ini dapat dilakukan dengan mengimplementasikan fungsi umpan balik (*feedback*), f(x) dan g(x), serta fungsi *filter*, h(x), beberapa kali. Untuk meneyederhanakan implementasi ini, 15 bit terakhir dari *shift register*, s_i, 65 <= i <= 79 dan b_i, 65 <= i <= 79 tidak digunakan pada fungsi *feedback* atau pada masukan di fungsi *filter*. Hal ini menyebabkan kecepatan dapat dengan mudah ditingkatkan sampai dengan 16 kali jika perangkat keras memungkinkan. Contoh implementasi jika kecepatan ditingkatkan dua kali lipat dapat ditunjukkan dengan gambar berikut:



Gambar 3 *Grain Cipher* dengan kecepatan dua kali lipat

Tentu saja *shift register* juga perlu diimplementasikan agar tiap bit digeser sejauh *t* langkah, bukan 1 langkah, ketika kecepatan ditingkatkan sebesar faktor *t*. Dengan meningkatkan kecepatan dengan faktor 16, keluaran *cipher* menjadi 16 bit/clock. Karena, pada inisialisasi kunci cipher di-clock 160 kali, kemungkinan peningkatan kecepatan terbatas ke faktor <= 16 yang dapat dibagi 160. Dan dengan begitu, jumlah *clock* yang digunakan pada inisialisasi kunci menjadi 160/t. Karena fungsi *filter* dan *feedback* kecil, maka peningkatan *throughput* dapat dilakukan dengan cara ini.

2.4 Contoh Implementasi

Contoh implementasi *grain cipher* dengan menggunakan bahasa VHDL adalah sebagai berikut:

```
architecture behav of grain is
signal lfsr, nfsr : unsigned(0 to 79);
signal func_h, func_g, func_f : std_logic;
begin
  -- outputs:
  H_O <= func_h;

  func_h <=
    nfsr(1) xor nfsr(2) xor nfsr(4) xor
nfsr(10) xor nfsr(31) xor nfsr(43) xor
nfsr(56) xor
    lfsr(25) xor nfsr(63) xor
    (lfsr(3) and lfsr(64)) xor
    (lfsr(46) and lfsr(64)) xor
    (lfsr(64) and nfsr(63)) xor
    (lfsr(3) and lfsr(25) and lfsr(46)) xor
    (lfsr(3) and lfsr(46) and lfsr(64)) xor
    (lfsr(3) and lfsr(46) and nfsr(63)) xor
    (lfsr(25) and lfsr(46) and nfsr(63))
xor
    (lfsr(46) and lfsr(64)and nfsr(63));

  func_g <=
    lfsr(0) xor
    nfsr(62) xor nfsr(60) xor nfsr(52) xor
nfsr(45) xor nfsr(37) xor nfsr(33) xor
nfsr(28) xor nfsr(21) xor nfsr(14) xor
nfsr(9) xor nfsr(0) xor
    (nfsr(63) and nfsr(60)) xor
    (nfsr(37) and nfsr(33)) xor
    (nfsr(15) and nfsr(9)) xor
```

```
    (nfsr(60) and nfsr(52) and nfsr(45))
xor
    (nfsr(33) and nfsr(28) and nfsr(21))
xor
    (nfsr(63) and nfsr(45) and nfsr(28) and
nfsr(9)) xor
    (nfsr(60) and nfsr(52) and nfsr(37) and
nfsr(33)) xor
    (nfsr(63) and nfsr(60) and nfsr(21) and
nfsr(15)) xor
    (nfsr(63) and nfsr(60) and nfsr(52) and
nfsr(45) and nfsr(37)) xor
    (nfsr(33) and nfsr(28) and nfsr(21) and
nfsr(15) and nfsr(9)) xor
    (nfsr(52) and nfsr(45) and nfsr(37) and
nfsr(33) and nfsr(28) and nfsr(21));

  func_f <=
    lfsr(62) xor lfsr(51) xor lfsr(38) xor
lfsr(23) xor lfsr(13) xor lfsr(0);

  -- the shift registers:
  sr_proc : process(CLK_I)
  begin
    if rising_edge(CLK_I) then
      if CLKEN_I = '1' then
        lfsr <= lfsr sll 1;
        nfsr <= nfsr sll 1;
        if INJECT_INPUT_I = '1' then
          lfsr(79) <= IV_I or PAD_IV_I;
          nfsr(79) <= KEY_I;
        else
          lfsr(79) <= func_f xor
(ADD_OUTPUT_I and func_h);
          nfsr(79) <= func_g xor
(ADD_OUTPUT_I and func_h);

        end if;
      end if;
    end if;
  end process;

end behav;
```

3. Serangan Pada *Grain Cipher*

Pada bagian ini akan dibahas beberapa serangan yang umum pada stream cipher dan analisis dari seberapa besar pengaruh mereka pada grain cipher.

3.1 *Correlation Attack*

Correlation attack adalah serangan yang termasuk pada kategori *known plaintext attacks* untuk menyerang *stream cipher* yang memiliki *keystream* yang dibangkitkan dengan cara kombinasi keluaran beberapa LFSR menggunakan fungsi boolean. *Correlation attack* memanfaatkan kelemahan statistik yang muncul dikarenakan pemilihan fungsi boolean yang buruk.

Pada *grain cipher*, karena sifat statistik panjang maksimum sekuens LFSR, bit-bit dalam LFSR relatif seimbang. Tapi hal ini mungkin tidak terjadi pada NFSR xored. Namun, karena umpan balik g(x) di-xor dengan *state* pada LFSR, bit-bit dalam NFSR seimbang. Selain itu, g adalah fungsi seimbang. Oleh karena itu, bit dalam NFSR dapat diasumsikan tidak memiliki korelasi ke bit-bit LFSR.

Fungsi h dipilih untuk menjadi "kebal" korelasi terhadap urutan pertama. Hal ini tidak menghalangi adanya korelasi dari output h (x) ke jumlah input.

As one input comes from the NFSR and as h(x) is xored with a state bit of the Ketika salah satu masukan yang berasal dari NFSR dan ketika h(x) di-xor dengan bit *state* pada NFSR, korelasi output dari generator untuk jumlah bit LFSR akan begitu kecil sehingga tidak akan dapat dimanfaatkan oleh *correlation attack*.

3.2 *Algebraic Attack*

Prinsip dari *algebraic attack* adalah mencari persamaan pada cipher dengan bit kunci tidak diketahui, lalu mengisi pengubah-pengubah yang diketahui dan menyelesaikan persamaan tersebut. Maslah yang dihadapi oleh tipe serangan ini adalah persamaan tidak linear dengan derajat yang tinggi.

Pada grain cipher, pembangkit filter dengan fungsi keluaran h(x) dengan derajat berjumlah hanya tiga akan sangat rentan pada serangan aljabar. Di lain pihak, serangan aljabar tidak akan dapat bekerja untuk menyelesaikan state awal 160 bit pembangkit yang penuh, karena fungsi update pada NFSR tidak linear, dan state berikutnya dari NFSR sebagai fungsi dari initial state akan memiliki derajat aljabar yang bervariasi namun besar.

Dengan menggunakan inisialisasi kunci, dimungkinkan untuk menyatakan keluaran pembangkit sebagai fungsi dari bit state LFSR saja. Karena fungsi filter h(x) memiliki satu masukan yang berasal dari NFSR, dan h(x) akan di-xor dengan bit state NFSR, derajat aljabar dari bit keluaran ketika dinyatakan sebagai fungsi dari LFSR akan menjadi besar dan bervariasi. Hal ini akan mengalahkan serangan aljabar (*algebraic attack*).

3.3 *Chosen IV Attack*

Kondisi yang diperlukan untuk mengalahkan *chosen-IV attack* adalah bahwa untuk dua IV berbeda yang dihasilkan tidak memiliki hubugan baik secara aljabar maupun statistik. Jumlah siklus pada inisialisasi kunci di *grain cipher* telah dibuat sehingga dua IV berbeda yang dihasilkan hampir acak. Hal ini seharusnya dapat mencegah *chosen-IV attack*.

3.4 *Fault Attack*

Fault attack adalah salah satu serangan yang paling kuat pada stream cipher. Misalkan diasumsikan bahwa penyerang dapat mengganti beberapa beberapa bit flipping yang salah pada

salah satu dari dua *feedback register* yang ada. Tetapi dia hanya memiliki sebagian control untuk angka, posisi, dan waktu yang tepat. Asumsi yang lebih jauh lagi adalah bahwa penyerang dapat membalik (*flip*) satu bit pada satu waktu dan posisi, yang dia tidak tahu perrsis. Selain itu, dia dapat mengatur ulang perangkat ke keadaan semula dan kemudian melakukan perubahan secara acak ke perangkat. Asumsi terkuat adalah bahwa penyerang dapat mengubah satu bit pada LFSR dan dia dapat mengetahui dengan tepat posisinya.

Selama perbedaan yang dihasilkan oleh bit-bit yang berada di LFSR tidak terpropagasi ke posisi bi+63, perbedaan yang muncul pada keluaran cipher hanya datang dari masukan h(x) dari LFSR.

Jika penyerang dapat mengatur ulang perangkat ke keadaan semula (*reset*) dan mengubah satu bit berulang-ulang pada posisi yang berbeda sehingga dia dapat menebak dari perbedaan keluaran, tidak dapat dihalangi bahwa informasi bagian dari bit state pada LFSR dapat diambil. Tapi pada asumsi yang lebih realistik bahwa posisi bit yang terubah tidak dapat diketahui, serangan akan sulit untuk dilakukan karena akan lebih sulit untuk menganalisis dari perbedaan yang muncul pada keluaran.

Perubahan yang terjadi juga dapat terjadi pada NFSR saja. Pada kasus ini, perubahan tidak mempengaruhi isi dari LFSR. Hal ini dikarenakan isi NFSR terpropagasi secara tidak linear dan perubahannya akan jauh lebih sulit untuk diprediksi. Oleh karena itu, serangan pada NFSR akan lebih sulit untuk dilakukan.

4. PERBANDINGAN

Pada bagian ini akan diperlihatkan penjelasan umum tentang *cipher* yang bersifat *hardware oriented* lainnya, yaitu *A5/1* dan *Trivium*, serta perbandingan *Grain Cipher* dengan *cipher-cipher* tersebut.

4.1 A5/1

A5/1 adalah *stream cipher* yang digunakan untuk menyediakan keamanan komunikasi pada standar telepon selular GSM. A5/1 digunakan di Eropa dan Amerika Serikat. Cipher ini dikembangkan pada tahun 1987 ketika GSM masih belum digunakan untuk luar Eropa. Cipher ini pada awalnya dirahasiakan, namun akhirnya desain umumnya tersebar ke masyarakat luas pada tahun 1994, dan keseluruhan algoritmanya terbuka dengan cara reverse engineering oleh Marc Briceno dari telepon GSM. Akan tetapi, sampai tahun 2000 masih sekitar 130 juta pengguna GSM bergantung pada cipher A5/1 ini untuk kerahasiaan dalam komunikasi suara.

Transmisi GSM diorganisasikan sebagai sekuens dari *bursts*. Pada channel pada umumnya dan secara satu arah, satu *burst* dikirimkan setiap 4.615 milidetik dan mengandung 114 bit data yang tersedia untuk informasi. A5/1 digunakan untuk menghasilkan 114 bit sekuens keystream untuk setiap burst yang di-xor dengan 114 bit sebelum modulasi. A5/1 diinisialisasi menggunakan kunci 64-bit bersama-sama dengan nomor *frame* sebesar 22 bit. Pada implementasinya di GSM, sepuluh dari bit kunci ditetapkan dengan 0, sehingga panjang kunci efektif adalah 54 bit. A5/1 juga dapat digunakan untuk enkripsi data pada EDGE, dengan sampai delapan burst yang dikirimkan setiap 4.615 milidetik, masing-masing mengandung data 348 bit.

A5/1 didasarkan pada kombinasi dari 3 LFSR dengan *clocking* tidak tetap. Ketiga shift register tersebut adalah sebagai berikut:

Tabel 1 Shift Register pada A5/1 Cipher

LSFR	Panjang (bit)	Karakteristik Polinomial	Clocking Bit
1	19	$x^{18}+x^{17}+x^{16}+x^{13}+1$	8
2	22	$x^{21}+x^{20}+1$	10
3	23	$x^{22}+x^{21}+x^{20}+x^{7}+1$	10

Indeks pada bit memiliki representasi dengan indeks pada *least significant bit* (LSB) adalah 0. Register di-*clocked* dengan penggunaan aturan mayoritas. Setiap register memiliki clocking bit yang berasosiasi dengannya. Pada setiap siklus, clocking bit dari ketiga register diperiksa dan bit maoritas ditentukan. Register tersebut di-clock apabila clocking bit yang berasosiasi dengannya tersebut cocok dengan bit mayoritas. Dengan begitu, pada tiap langkah dua atau tida register di-clock, dan tiap reister melangkah dengan peluang ¾.

Pada awalnya, register diinisialisasi dengan 0. Selanjutnya untuk 64 siklus, kunci rahasia sebesar 64 bit dimasukkan berdasarkan aturan berikut: pada siklus 0 <= i <= 64, bit kunci ke-I ditambahkan ke LSB dari tiap register menggunakan xor. Setelah itu, register di-*clock*. Dengan cara yang hampir sama, 22 bit dari nomor frame ditambahkan dalam 22 siklus. Kemudian, keseluruhan system di-*clock* dengan menggunakan mekanisme *clocking* mayoritas untuk 100 siklus dengan keluaran diabaikan. Setelah hal ini selesai dilakukan, cipher akan siap untuk membuat dua sekuens 114 bit keystream keluaran, satu untuk *downlink* (jalur unduh), dan satu untuk uplink (jalur unggah).

4.2 Trivium

Trivium adalah *synchronous stream cipher* yang didesain oleh Christophe De Canniere dan Bart Preneel, dan diikutsertakan pada kompetisi eSTREAM, dan telah dipilih sebagai bagian dari *portfolio* untuk *cipher* yang berorientasi pada perangkat keras. *Cipher* ini membangkitkan sampai dengan 2^{64} bit keluaran dari kunci dengan panjang 80 bit dan IV dengan panjang 80 bit.

Trivium memiliki *state internal* sebesar 288 bit. State ini terdiri dari tiga *shift register* dengan panjang yang berbeda-beda. Pada setiap *round*, satu bit digeser ke masing-masing dari tiga *shift register* menggunakan kombinasi non-linear dari *tap* dan satu register lain; satu bit keluaran dihasilkan. Untuk menginisialisasi *cipher* ini, kunci dan initial value ditulis ke dua dari tiga *shift register* yang ada, dengan bit yang tersisa dimulai pada pola yang tetap; kemudian *state* dari *cipher* diperbaharui 4 x 288 = 1152 kali, sehingga setiap bit dari *state internal* bergantung pada setiap bit dari kunci dan IV secara tidak linear dan kompleks. Pada 64 bit pertama dari tiap shift register, tidak ada *tap* yang muncul, dengan begitu setiap bit state tidak digunakan sampai setidaknya 64 *round* setelah dibangkitkan.

Trivium dapat dispesifikasikan dengan menggunakan persamaan-persamaan berikut:

$$a_i = c_{i-66} + c_{i-111} + c_{i-110}\, c_{i-109} + a_{i-69}$$
$$b_i = a_{i-66} + a_{i-93} + a_{i-92}\, a_{i-91} + b_{i-78}$$
$$c_i = b_{i-69} + b_{i-84} + b_{i-83}\, b_{i-82} + c_{i-87}$$

Bit keluaran $r_0 \ldots r_2^{64}{}_{-1}$ dibangkitkan dengan

$$r_i = c_{i-66} + c_{i-111} + a_{i-66} + a_{i-93} + b_{i-69} + b_{i-84}$$

Dengan kunci 80 bit: $k_0 \ldots k_{79}$ dan IV l bit $v_0 \ldots v_{l-1}$ (dengan $0 \leq l \leq 80$), Trivium diinisialisasikan sebagai berikut:

$$(a_{-1245} \ldots a_{-1153}) = (0, 0 \ldots 0, k_0 \ldots k_{79})$$
$$(b_{-1236} \ldots b_{-1153}) = (0, 0 \ldots 0, v_0 \ldots v_{l-1})$$
$$(c_{-1263} \ldots c_{-1153}) = (1, 1, 1, 0, 0 \ldots 0)$$

Indeks negatif yang besar pada initial value merefleksikan bahwa perlu 1152 langkah sebelum keluaran dihasilkan.

4.3 Perbandingan

Perbandingan properti / fitur dasar dari ketiga *cipher* dapat ditunjukkan dengan tabel di bawah ini:

Tabel 2 Perbandingan Properti Dasar Grain, A5/1, dan Trivium Cipher

Cipher	Panjang Kunci (bits)	Panjang IV (bits)	Ukuran Internal State (bits)	Komponen Dasar
Grain	80	64	160	LFSR, NFSR, fungsi filter
A5/1	64	22	64	LFSR, unit pengontrol *clock*
Trivium	80	80	288	LFSR, NFSR

Grain cipher, Trivium, dan A5/1 adalah cipher yang berbasis pada LFSR dan NFSR. Pada arsitektur dasar, setiap shift register digeser sebesar satu posisi pada setiap *clock cycle*, dan hanya satu bit dari keystream yang dihasilkan pada satu waktu. Dengan begitu, *throughput* maksimum rangkaian sama dengan satu bit dibagi dengan perioda minimum clock dan jika direpresentasikan dalam Mbits/s adalah sama dengan frekuensi maksimum clock dalam MHz.

Kunci dan IV di-load satu bit tiap *clock cycle*, sehingga *key setup latency* sama dengan total dari panjang kunci dan IV, ditambahkan dengan jumlah *clock cycle* yang dibutuhkan untuk inisialisasi LFSR dan NFSR.

Unutk meningkatkan throughput, LFSR dan NFSR dapat digeser sebesar posisi *d* dalam satu waktu, dan arsitektur menghasilkan sebanyak *d* bit keystream tiap *clock cycle*. Arsitektur ini meningkatkan *throughput* dengan factor dekat ke nilai d, tapi pada waktu yang sama juga memiliki area yang lebih besar, karena semua *logic* umpan balik harus diulang sebanyak d kali.

Nilai maksimum dari d dapat ditentukan dari jarak minimum anatara entri serial dari tiap *shift register* dan posisi *tap* pertama yang digunakan di *logic* umpan balik. Dengan begitu, nilai d untuk masing-masing *cipher* berbeda dan untuk Grain: d=2, 4, 8, 16; untuk Trivium: d=2, 4, 8, 16, 32, 64; dan untuk A5/1: d=3, 4.

Hasil dari implementasi FPGA (berdasar referensi) dari ketiga *cipher* dapat direpresentasikan dengan tiga tabel di bawah ini. Perangkat keras yang digunakan berasal dari jenis Xilinx Spartan 3.

Tabel 3 Hasil Uji Performansi Grain Cipher

Parallelization factor *d*	Maximum clock frequency	Waktu minimum key setup untuk *k=d*		Maximum throughput		Area		Throughput to area ratio	
	MHz	cycles	ns	Mbit/s	x basic	CLB slices	x basic	Mbit/s / CLB slices	x basic
1 (*basic*)	193	304	1575	193	1.0	122	1.0	1.58	1.0
2	168	152	905	336	1.7	147	1.2	2.29	1.4
4	170	76	447	680	3.5	173	1.4	3.93	2.5
8	161	38	236	1288	6.7	244	2.0	5.28	3.3
16	155	19	123	2480	12.8	356	2.9	6.97	4.4

Tabel 4 Hasil Uji Performansi A5/1 Cipher

Parallelization factor *d*	Maximum clock frequency	Waktu minimum key setup untuk *k=d*		Maximum throughput		Area		Throughput to area ratio	
	MHz	cycles	ns	Mbit/s	x basic	CLB slices	x basic	Mbit/s / CLB slices	x basic
1 (*basic*)	174	186	1069	174	1.0	57	1.0	3.05	1.0
3	114	63	553	342	2.0	142	2.5	2.41	0.8
4	79	47	595	316	1.8	287	5.0	1.10	0.4

Tabel 5 Hasil Uji Performansi Trivium Cipher

Parallelization factor *d*	Maximum clock frequency	Waktu minimum key setup untuk *k=d*		Maximum throughput		Area		Throughput to area ratio	
	MHz	cycles	ns	Mbit/s	x basic	CLB slices	x basic	Mbit/s / CLB slices	x basic
1 (*basic*)	201	1312	6527	201	1.0	188	1.00	1.07	1.00
2	202	656	3248	404	2.0	189	1.01	2.14	2.00
4	203	328	1616	812	4.0	199	1.06	4.08	3.82
8	193	16	85	15	7.	19	1.	7.	7.
		4	0	44	7	9	06	76	26
16	191	82	429	3056	15.2	227	1.21	13.46	12.59
32	202	41	203	6464	32.2	264	1.40	24.48	22.90
64	190	21	108	12160	60.5	388	2.06	31.34	29.31

Hasil yang dapat dilihat adalah *maximum clock frequency* dalam MHz, *maximum encryption/decryption throughput* dalam Mbit/s, area dalam the jumlah *CLB(configurable logic blocks) slices*, and rasio *throughput* dan *area*.

Selain itu juga ditampilkan waktu minimum *key setup* yang termasuk waktu load kunci dan IV serta waktu operasi inisialisasi tambahan lainnya.

Parameter d adalah parallelization factor yang menentukan jumlah bit *keystream* yang dihasilkan tiap *clock cycle*. Parameter *k*, yang merupakan jumlah bit dari kunci dan IV yang di-*load* ke *state internal* tiap *clock cycle*, dipilih untuk sama dengan nilai *d*. Dengan begitu, peningkatan *throughput* disertai dengan penurunan pada waktu *key setup*.

Untuk *maximum throughput, area,* dan rasio *throughput to area*, dipelihatkan perubahan relatif terhadap nilai pada arsitektur dasar (d=1). Dapat dilihat bahwa terjadi peningkatan terbesar pada *maximum throughput* yang dimungkinkan pada Trivium. Pada cipher ini, dengan *parallelization factor d*=64, *throughput* meningkat sampai dengan 60.5 kali, dan rasio *throughput to area* meningkat sampai 29.31 kali. Peningkatan ini jauh lebih kecil pada Grain dan A5/1, dan pada A5/1 lebih pada hanya *throughput* bukan rasio *throughput to area*. Tetapi, meskipun begitu, *maximum throughput* dari *grain cipher* masih tetap jauh di atas *A5/1 cipher*.

Perbandingan dari ketiga *cipher* ini dengan optimisasi pada rasio *maximum throughput to area* dapat diperlihatkan dengan tabel di bawah ini:

Tabel 6 Perbandingan dengan Optimisasi pada Rasio *Maximum Throughput to Area*

Cipher	Maximum clock frequency	Waktu minimum key setup untuk *k=d*		Maximum throughput		Area		Throughput to area ratio	
	MHz	cycles	ns	Mbit/s	Trivium /cipher	CLB slices	Cipher / Trivium	Mbit/s / CLB slices	Trivium /Cipher
Trivium (d=k=6	190	21	108	12160	1.0	388	1.00	**31.34**	**1.0**

4)									
Grain (d=k=16)	155	19	123	2480	4.9	356	0.92	**6.97**	**4.5**
A5/1 (d=k=1)	174	186	1069	174	69.9	57	0.15	**3.05**	**10.3**

Pada tabel dapat dilihat bahwa Trivium cipher memiliki rasio *throughput to area* terbesar. Sedangkan dari segi area minimum, A5/1 memiliki nilai terkecil. Tapi, A5/1 merupakan cipher yang telah lama bisa dipecahkan dan dengan begitu berarti kurang aman. Dari segi maximum *throughput to area*, *grain cipher* tidak memiliki perbedaan yang cukup jauh dengan Trivium, dan area yang digunakan oleh *grain cipher* lebih sedikit, yang berarti kompleksitas dari *cipher* ini lebih rendah.

5. Kesimpulan

Grain cipher adalah *cipher* yang berorientasi pada perangkat keras.

Grain cipher terdiri dari tiga bagian utama, yaitu sebuah LFSR, sebuah NFSR, dan sebuah fungsi penyaring (*filter)*.

Throughput dari *grain cipher* dapat ditingkatkan berbanding lurus dengan peningkatan kemampuan perangkat keras.

Untuk ketiga cipher yang bersifat *hardware-oriented* dan berbasis pada LFSR dan NFSR, yaitu *grain cipher*, *A5/1 cipher,* dan *Trivium cipher*, *throughput* tertinggi dihasilkan oleh Trivium, dan area minimum adalah oleh A5/1 cipher. Tetapi, *grain cipher* memiliki *throughput* yang tidak berbeda jauh dengan Trivium dan area yang digunakan lebih sedikit. Hal ini mengindikasikan *grain cipher* cukup baik dan juga memiliki kompleksitas perangkat keras yang lebih kecil dari *Trivium cipher*.

DAFTAR PUSTAKA

[1] Gaj, Kris. Comparison of hardware performance of selected Phase II eSTREAM candidates. *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.153&rep=rep1&type=pdf.* Tanggal akses: 3 Maret 2010 pukul 20:05.

[2] Gurkaynak, Frank K. Hardware Evaluation of eSTREAM Candidates. *http://asic.ethz.ch/estream/sasc2006_talk.pdf.* Tanggal akses: 3 Maret 2010 pukul 20:05.

[3] Hell, Martin. Grain - A Stream Cipher for Constrained Environments. *http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf* . Tanggal akses: 3 Maret 2010 pukul 20:20.

[4] Hwang, David. Comparison of FPGA-Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. *http://volgenau.gmu.edu/~kgaj/publications/conferences/GMU_SASC_2008.pdf.* Tanggal akses: 3 Maret 2010 pukul 23:42.

[5] A5/1. http://en.wikipedia.org/wiki/A5/1. Tanggal akses: 24 Maret 2010 pukul 17:15.

[6] Trivium (cipher). http://en.wikipedia.org/wiki/Trivium_%28cipher%29. Tanggal akses: 24 Maret 2010 pukul 17:25.