Studi Kriptanalis Algoritma ORYX

Neni Adiningsih – NIM : 13503007

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13007a@students.if.itb.ac.id

Abstrak

Pada makalah akan dibahas mengenai algoritma ORYX. ORYX merupakan stream cipher sederhana yang berbasis pada *binary linear feedback shift registers* (LFSRs)[1]. Stream cipher merupakan algoritma kriptografi yang beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal yang dalam hal ini rangkaian bit dienkripsi/dideskripsi bit per bit.

Algoritma ini digunakan untuk melindungi data selama proses transmisi. Cipher ORYX digunakan sebagai *keystream generator* yang menghasilkan byte-byte random. Enkripsi dapat dilakukan dengan melakukan operasi XOR antara byte-byte keystream yang dihasilkan dengan byte-byte data. Sedangkan deskripsi dilakukan dengan melakukan operasi XOR antara byte-byte keystream dengan ciphertext. Dengan mengetahui ciphertext dan plaintext maka segmen *keystream* juga dapat diketahui.

Serangan yang dilakukan terhadap algoritma stream cipher ORYX adalah menggunakan 25-27 byte *known plaintext* dan membutuhkan kompleksitas waktu sampai 216. Dengan tanpa memperhatikan time schedule, serangan ini dengan langsung dapat menghasilkan 96 bit *initial state* dari ORYX. Kemudian teknik ini diperluas untuk menyerang ORYX dengan *Ciphertext only Attack*. Melalui hasil ini dapat disimpulkan bahwa kebanyakan komunikasi mobile masih menggunakan algoritma enkripsi yang dalam segi keamanannya kurang kuat. Tetapi untuk ke depannya Code-Division Multiple Access (CDMA) tidak lagi menggunakan algoritma ini sebagai algoritma enkripsinya, yaitu menggunakan algoritma AES dengan mode stream cipher.

***Kata kunci**: CDMA, ORYX, stream cipher, LFSR, security, enkripsi, dekripsi.*

1. Pendahuluan

Komunikasi menggunakan telepon seluler berkembang sangat pesat pada akhir dekade ini. Komunikasi seluler dikirimkan melalui link radio sehingga sangat mudah bagi orang yang tidak bertanggung jawab untuk menyadap/mendapatkan informasi secara diam-diam dari sistem tanpa terdeteksi.

Pengiriman data melalui media elektronik memerlukan suatu proses yang dapat menjamin keamanan dan keutuhan dari data yang dikirimkan tersebut. Data tersebut harus tetap rahasia selama pengiriman dan harus tetap utuh pada saat penerimaan di tujuan. Untuk memenuhi hal tersebut, dilakukan proses penyandian (enkripsi dan dekripsi) terhadap data yang akan dikirimkan.

Enkripsi dilakukan pada saat pengiriman dengan cara mengubah data asli menjadi data rahasia sedangkan dekripsi dilakukan pada saat penerimaan dengan cara mengubah data rahasia menjadi data asli. Jadi data yang dikirimkan selama proses pengiriman adalah data rahasia,

sehingga data asli tidak dapat diketahui oleh pihak yang tidak berkepentingan. Data asli hanya dapat diketahui oleh penerima dengan menggunakan kunci rahasia.
Oleh karena itu digunakan algortima kriptografi untuk membangun sebuah lingkungan komunikasi seluler yang aman, yaitu dapat menjaga kepentingan individu/kelompok dan mencegah kecurangan yang dapat terjadi.

Device generasi pertama komunikasi seluler masih bersifat analog. Pada telepon seluler analog sangat jarang digunakan enkripsi dan beberapa device yang menggunakan enkripsi tingkat keamanannya pun masih sangat rendah. Hal ini berbeda dengan komunikasi seluler digital yang saat ini sedang berkembang yaitu GSM dan TIA (TDMA dan CDMA).

Pada sistem komunikasi digital tingkat keamanannya lebih tinggi dan telah menggunakan algoritma enkripsi modern untuk pengiriman data. Namun algoritma yang digunakan bukanlah algoritma yang benar-benar robust. Misalnya pada telepon GSM[4], memungkinkan seseorang dapat memecahkan algoritma voice privacy yang digunakan yaitu A5 cipher dengan menggunakan plaintext yang diketahui. Sedangkan pada CDMA terdapat beberapa algoritma kriptografi primitif yang digunakan, diantaranya[2]:

a. CAVE, digunakan untuk challenge-response authentication protocols dan key generation.

b. ORYX, LFSR-based stream cipher untuk wireless data services.

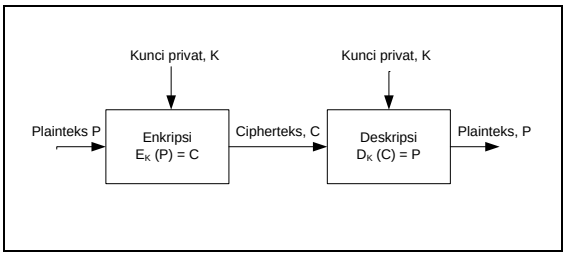
c. CMEA, block cipher sederhana yang digunakan utnuk enkripsi data pada channel.

d. Pada *voice privacy*, sistem TDMA menggunakan XOR mask atau teknik keyed spread spectrum yang dikombinasikan dengan LFSR mask.

Pada algoritma *Voice Privacy* yang digunakan oleh sistem TDMA dapat dikatakan rentan karena algoritma ini menggunakan XOR *mask* yang diulang. Beberapa sistem dapat dengan mudah diserang menggunakan *ciphertext alone*[7]. Algoritma CMEA rentan terhadap *known plaintext attack*[8]. Sedangkan algoritma ORYX akan dibahas dalam makalah ini.

2. Tipe dan Mode Algoritma Simetri

Sebuah algoritma dikatakan sebagai algoritma simetri jika kunci enkripsi sama dengan kunci deskripsinya.



Gambar 1 Skema Kriptografi Simetri

Algortima kriptografi simetri yang dalam bentuk bit dapat dibagi menjadi dua kategori, yaitu:

a. Cipher Aliran
   Algoritma kriptografi yang beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal yang dalam hal ini rangkaian bit dienkripsi/dideskripsi bit per bit.
   Seperti yang sudah dijelaskan di atas bahwa algoritma ORYX merupakan algoritma cipher aliran.

b. Cipher Block
   Algoritma kriptografi yang beroperasi pada plainteks/cipherteks dalam bentuk blok bit, yang dalam hal ini rangkaian bit dibagi menjadi blok-blok bit yang panjangnya sudah ditentukan sebelumnya.
   Misalnya panjang blok adalah 64 bit, maka algoritma memperlakukan 8 karakter dalam setiap kali penyandian.

2.1 Cipher Aliran

Cipher aliran mengenkripsi plainteks menjadi cipherteks bit per bit (1 bit setiap kali transformasi). Pada cipher aliran, bit hanya mempunyai dua buah nilai yaitu berubah atau tidak berubah. Nilai ini ditentukan oleh kunci enkripsi yaitu *keystream*.
Aliran bit kunci dibangkitkan oleh *keystream generator* yang kemudian diXORkan dengan bit plainteks untuk menghasilkan bit cipherteks.

$$c_i = p_i \oplus k_i$$

Sedangkan untuk proses deskripsi bit-bit cipherteks di-XOR-kan dengan aliran bit kunci,

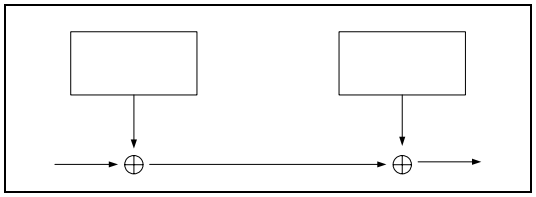
keystream, yang sama untuk menghasilkan bit-bit plainteks:

$$p_i = c_i \oplus k_i$$

karena

$$c_i \oplus k_i = (p_i \oplus k_i) \oplus k_i = p_i \oplus (k_i \oplus k_i) = p_i \oplus 0 = p_i$$

sehingga dapat disimpulkan bahwa proses enkripsi dua kali berturut-turut dapat menghasilkan kembali plainteks semula.



Gambar 2 Konsep cipher aliran

Keamanan pada cipher aliran bergantung seluruhnya pada pembangkit aliran bit kunci. Jika pembangkit mengeluarkan aliran bit kunci yang seluruhnya nol (0), maka cipherteks akan sama dengan plainteks dan proses enkripsi akan tidak ada artinya.

Jika pembangkit mengeluarkan aliran bit kunci dengan pola 16-bit yang berulang, maka algoritma enkripsi menjadi sama seperti dengan XOR sederhana yang memiliki tingkat keamanan yang tidak berarti.

Jika pembangkit mengeluarkan aliran bit kunci yang benar-benar acak (*truly random*), maka algoritma enkripsinya sama dengan *one time pad* dengan tingkat keamanan yang sempurna (aliran bit kunci sama dengan panjang plainteks) sehingga menghasilkan cipher aliran yang *unbreakable cipher*.

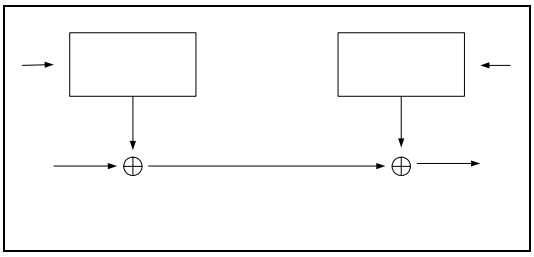
Tingkat keamanan cipher aliran sangat tergantung pada *keystream* yang dihasilkan, semakin acak output yang dihasilkan oleh *keystream generator* semakin sulit kriptanalis memecahkan cipherteks tersebut.

2.2 Keystream Generator

Pembangkit aliran bit kunci dapat membangkitkan bit-bit kunci (*keystream*) berbasis bit per bit atau dalam bentuk blok-blok bit. Untuk yang terkahir ini, cipher blok dapat digunakan untuk memperoleh cipher aliran.

Keystream generator diimplementasikan sebagai prosedur algoritmik, sehingga bit-bit kunci (*keystream*) dapat dibangkitkan dibangkitkan secara simultan oleh pengirim dan penerima pesan.

Prosedur algoritmik tersebut menerima masukan sebuah kunci U. Keluaran dari prosedur merupakan fungsi dari U. Pembangkitkan harus menghasilkan bit-bit kunci yang kuat secara kriptografi.



Gambar 3 Cipher aliran dengan pembangkit aliran bit kunci yang bergantung pada kunci U

Kunci U yang dihasilkan relatif pendek untuk membangkitkan bit-bit yang panjang. Secara umum, jika panjang kunci U adalah n bit, maka bit-bit kunci tidak akan berulang sampai 2^n-1 bit.

U merupakan besaran konstan sehingga aliran bit-bit kunci yang dihasilkan pada setiap lelaran tidak berubah jika bergantung hanya pada U. Hal ini berarti bahwa pembangkit aliran bit kunci tidak boleh mulai dengan kondisi awal yang sama supaya tidak menghasilkan kembali bit-bit kunci yang sama pada setiap lelaran. Oleh karena itu beberapa pembangkit aliran bit kunci menggunakan *seed* (umpan) yang disimbolkan oleh Z (Initial Vector/IV) agar diperoleh kondisi awal yang berbeda pada setiap lelaran.

Dengan demikian, bit-bit kunci K dapat dinyatakan sebagai hasil dari fungsi *g* dengan parameter kunci U dan masukan Z:

$$K = g_U(Z)$$

sehingga proses enkripsi dan deskripsi didefinisikan sebagai:

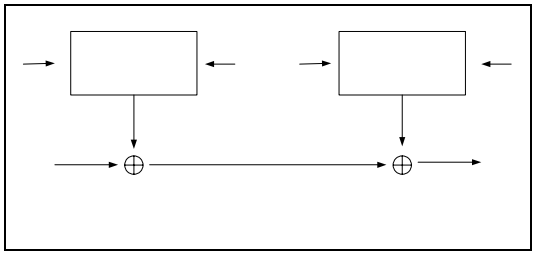
$$C = P \oplus K = P \oplus g_U(Z)$$

$$P = C \oplus K = C \oplus g_U(Z)$$

Dengan nilai Z yang berbeda-beda pada setiap lelaran menghasilkan bit-bit kunci yang berbeda pula.

Menggunakan pasangan Z dan U yang sama dua kali dapat menyebabkan bit-bit kunci yang sama setiap kali. Penggunaan *keystream* yang sama dua kali memudahkan jenis penyerangan *cipher attack*.

Karena bit-bit kunci hanya bergantung pada Z dan U, maka bit-bit kunci ini tidak terpengaruh oleh kesalahan transmisi di dalam cipherteks. Jadi, kesalahan 1-bit pada transmisi cipherteks hanya menghasilkan kesalahan 1-bit pada plainteks hasil deskripsi.



Gambar 4 Cipher aliran dengan pembangkit aliran bit kunci yang bergantung pada kunci U dan umpan (*seed*) Z

Karena pengirim dan penerima harus menghasilkan bit-bit kunci yang sama maka keduanya harus memiliki kunci U yang sama.

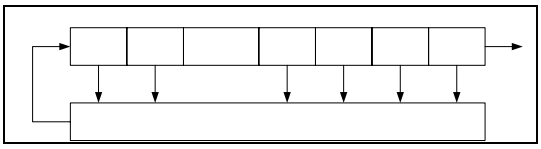
2.3 Linier Feed Back Shift Register

Konsep register geser (*shift register*) banyak diterapkan pada bidang kriptografi maupun teori pengkodean. Algoritma cipher aliran (*stream cipher*) misalnya, banyak yang didasarkan pada register geser. LFSR adalah salah satu contoh sederhana penggunaan register geser sebagai pembangkit aliran bit kunci acak.

Register geser umpan balik (*feedback shift register*) atau FSR terdiri dari dua bagian:

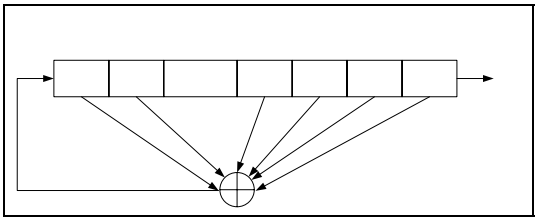
a. Register geser
   Yaitu barisan bit-bit ($b_n b_{n-1}.....b_4 b_3 b_2 b_1$) yang panjangnya *n* (disebut juga register geser *n*-bit).

b. Fungsi umpan balik
   Yaitu fungsi yang menerima masukan dari register geser dan mengembalikan nilai fungsi ke register geser.



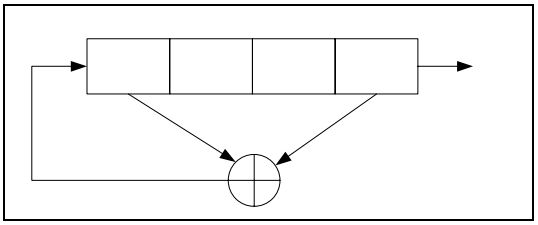
Gambar 5 Bagian-bagian FSR

Setiap kali 1 bit dibutuhkan, semua bit di dalam register digeser 1 bit ke kanan. Bit paling kiri (b_n) dihitung sebagai fungsi bit-bit lain di dalam register tersebut. Keluaran dari register geser adalah 1 bit (yaitu bit yang tergeser). Periode register geser adalah panjang barisan keluaran sebelum ia berulang kembali.



Gambar 6 LFSR Sederhana

Gambar di atas merupakan contoh dari register geser umpan balik (feedback shift register) yaitu Linier Feedback Shift Register (LFSR). Fungsi umpan balik adalah peng-XOR-an bit-bit tertentu dalam register.



Gambar 7 LFSR 4-bit

Gambar di atas adalah LFSR 4-bit yang dalam hal ini fungsi umpan balik meng-XOR-kan b_4 dengan b_1 dan menyimpan hasilnya di b_4.

$$b_4 = f(b_1, b_4) = b_1 \oplus b_4$$

Jika register diinisialisasi dengan 1111, maka isi register (menyatakan status atau *state*) dan bit keluaran sebelum berulang kembali adalah:

Pengirim

L Keystream Generator Z

Penerima

Z Keystream Generator L

Keystream k

Keystream k

i	Isi Register	Bit Keluaran
0	1 1 1 1	
1	0 1 1 1	1
2	1 0 1 1	1
3	0 1 0 1	1
4	1 0 1 0	1
5	1 1 0 1	0
6	0 1 1 0	1
7	0 0 1 1	0
8	1 0 0 1	1
9	0 1 0 0	1
10	0 0 1 0	0
11	0 0 0 1	0
12	1 0 0 0	1
13	1 1 0 0	0
14	1 1 1 0	0

Tabel 1 Contoh Register LFSR

Barisan bit-bit keluaran (merupakan bit-bit acak) adalah:

1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 ....

LFSR *n*-bit mempunyai 2^n-1 status internal (keadaan isi register). Ini berarti, secara teoritis LFSR dapat membangkitkan 2^n-1 barisan bit acak-semu (*pseudo random*) sebelum perulangan. Jadi periode maksimal LFSR 2^n-1.

Disebutkan 2^n-1 karena bit 0 0 0 0 tidak berguna karena akan menghasilkan bit 0 yang tidak akan berakhir.

2.4 Serangan Terhadap Cipher Aliran

Serangan terhadap cipher aliran dapat dilakukan dengan beberapa cara, yaitu:

a. Known Plain Attack

Beberapa pesan yang formatnya terstruktur membuka peluang kepada kriptanalis untuk menerka plainteks dari cipherteks yang bersesuaian.

Jika seorang kriptanalis mempunyai potongan plainteks dan cipherteks yang berkoresponden, maka bagian bit aliran kunci yang berkoresponden dapat diketahui dengan meng-XOR-kan bit plainteks dengan bit cipherteks.

$$\begin{aligned} P \oplus C &= P \oplus (P \oplus K) \\ &= (P \oplus P) \oplus K \\ &= 0 \oplus K \\ &= K \end{aligned}$$

b. Ciphertext only Attack

Kriptanalis memiliki beberapa cipherteks dari beberapa pesan, semuanya dienkripsi dengan algoritma yang sama.

Serangan ini terjadi jika *keystream* yang sama digunakan dua kali terhadap potongan plainteks yang berbeda (*keystream reuse attack*). Misal, jika kriptanalis memiliki potongan cipherteks berbeda C1 dan C2 yang dienkripsi dengan bit-bit kunci yang sama kemudian diberlakukan operasi XOR pada keduanya maka akan dihasilkan plainteks yang ter-XOR-kan satu sama lain:

$$\begin{aligned} C1 \oplus C &= (P1 \oplus K) \oplus (P2 \oplus K) \\ &= (P1 \oplus P2) \oplus (K \oplus K) \\ &= (P1 \oplus P2) \oplus 0 \\ &= (P1 \oplus P2) \end{aligned}$$

Jika salah satu dari plainteks P1 atau P2 dapat diketahui atau diterka, maka dengan mengXORkan salah satu plainteks tersebut dengan cipherteksnya akan didapatkan bit-bit kunci yang berkoresponden:

$$P1 \oplus C1 = P1 \oplus (P1 \oplus K) = K$$

Sehingga P2 bisa didapatkan dengan memanfaatkan kunci K yang telah dihasilkan.

Akan tetapi jika kedua plainteks tidak diketahui maka dua plainteks yang ter-XOR tersebut dapat diketahui menggunakan nilai statistik dari pesan.

Untuk mengatasinya pengguna cipher aliran harus mempunyai bit-bit kunci yang tidak dapat diprediksi sehingga jika sebagian dari bit-bit kunci diketahui tetapi tidak memungkinkan bagi kriptanalis dapat mendeduksi bagian yang lain.

c. Flip-bit Attack

Serangan tipe ini tidak bertujuan untuk menemukan kunci atau mengungkap plainteks dari cipherteks tetapi bertujuan untuk mengubah bit cipherteks tertentu sehingga hasil deskripsinya berubah. Pengubahan dilakukan dengan membalikkan (*flip*) bit tertentu (0 menjadi 1, atau 1 menjadi 0).
Pengubah pesan tidak perlu mengetahui kunci, hanya saja perlu mengetahui posisi pesan yang ingin diubah saja.

Serangan ini memanfaatkan karakteristik cipher aliran yang sudah disebutkan di atas, bahwa kesalahan 1-bit pada cipherteks hanya menghasilkan kesalahan 1-bit pada plainteks hasil deskripsi.

3. Enkripsi pada Sistem Wireless

3.1 Secure Tunnels

Pada *free space*, tidak ada batas fisik yang dapat digunakan untuk mencegah intruder bisa mengambil atau menggunakan data secara diam-diam, hal ini juga terjadi pada komunikasi secara wireless.

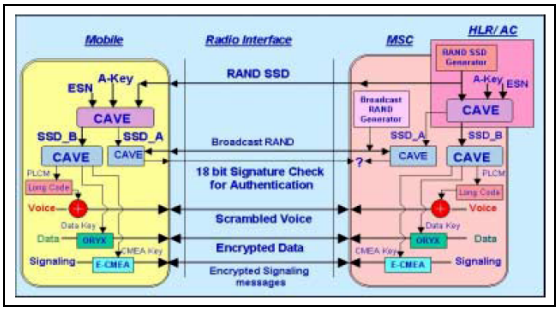
Oleh karena itu keamanan pada komunikasi wireless digunakan untuk melindungi data selama proses transmisi terhadap gangguan-gangguna yang tidak diinginkan dan digunakan untuk memastikan bahwa data hanya bisa dibaca atau digunakan oleh orang yang dituju saja. Hal ini dapat dilakukan melalui *secure tunnel*. Terdapat banyak cara untuk membentuk *secure tunnels* bagi komunikasi wireless. Salah satunya dengan menerapkan operasi kriptografi, yaitu untuk bangunan dasar dalam pembentukan *secure tunnels*.

a. User Authentication:
   Menentukan user yang telah disetujui.

b. Encryption:
   Melakukan enkripsi terhadap data sebelum data tersebut ditransmisikan dan mengirimkan data tersebut pada jalur yang dianggap efisien untuk proses deskripsi bagi penerima. Dengan enkripsi, user dapat mengirimkan informasi yang sangat sensitif relatif lebih 'aman'.

c. Message Authentication:
   Memastikan bahwa *messages* tidak dilakukan pengiriman berulang kali.

d. Access Control:
   Melakukan *Blocking* terhadap user yang tidak diinginkan untuk mengakses jaringan internal atau service.



Gambar 8 Otentifikasi dan mekanisme enkripsi pada CDMA.

3.2 Enkripsi untuk Sistem Wireless W-CDMA dan CDMA

Pada telekomiunikasi seluler terdapat beberapa tipe sistem sebelum berkembangnya dunia 3G. Pada 2G dan 2.5G, terdapat GSM dan GPRS yang digunakan di Eropa dan The Multiplexing Method yaitu Time Division Multiplexing Access (TDMA), dan IS-95 dan IS-98 yang mengembangkan Code Division Multiplexing Access (CDMA) sebagai *multiplexing method*.

Sistem CDMA digunakan di Amerika Utara kemudian sistem yang ada di Eropa mengalami perkembangan yang kemudian disebut sebagai UWTS yang mengalami modifikasi pada physical layer, wideband-CDMA (W-CDMA) yang menggunakan CDMA untuk access methods dan mengadopsi struktur *original network* GSM.

IS-95 and IS-98 berkembang menjadi CDMA2000 (IS-2000). Pada sistem GSMmenggunakan A5=1 sebagai algoritma enkripsinya. Algoritma ini digunakan untuk melindungi *voice message* yang melalui saluran udara. Pada W-CDMA, menggunakan block cipher Kusumi dengan mode stream cipher mode sebagai algoritma enkripsinya. Algoritma ini digunakan untuk melindungi informasi voice dan data. Sedangakan pada CDMA2000, menggunakan 3 LFSR yang berbeda dengan berdasarkan pada algoritma stream cipher yang digunakan untuk melindungi voice, data, dan signaling, secara berturut-turut.

Akan tetapi semua algoritma yang digunakan oleh sistem-sistem di atas telah dipecahkan. Untuk saat ini CDMA2000 mengimplementasikan algoritma AES Rijndael pada mode stream cipher.

3.2.1 W-CDMA

Enkripsi pada W-CDMA (UMTS) menggunakan mode stream cipher dengan meimplementasikan block cipher Kasumi.

3.2.2 CDMA

Code-Division Multiple Access (CDMA) merupakan teknologi seluler digital yang menggunakan teknik spectrum tersebar. CDMA memungkinkan banyak user untuk menggunakan jaringan/channel secara bersama.

Pada sistem CDMA percakapan individual disebar dengan *pseudo-random sequence*. CDMA secara konsisten menyediakan kapasitas yang lebih baik untuk komunikasi voice dan data daripada teknologi komersial lainnya.

Protokol keamanan jaringan CDMA berada pada sebuah 64-bit *authentication key* (A-Key) dan Electronic Serial Number (ESN) dari mobile.

Sebuah angka binary random yang disebut dengan RANDSSD yang dihasilkan pada HLR/AC juga berperan penting sebagai prosedur otentikasi. A-Key diprogram pada mobile nya dan disimpan di Authentication Center (AC) pada jaringan. Sebagai tambahan pada proses otentikasi, A-Key digunakan untuk menghasilkan *sub-keys* untuk *voice privacy* dan untuk enkripsi pesan.

CDMA menggunakan standar algoritma yaitu CAVE (Cellular Authentication and Voice Encryption) untuk menghasilkan sebuah 128-bit *sub-key* yang disebut dengan "*Shared Secret Data*" (SSD). A-Key, ESN dan *network-supplied* RANDSSD merupakan input bagi CAVE untuk menghasilkan SSD.

SSD terdiri dari 2 bagian, yaitu:

a. SSD_A (64 bit)
   Untuk membentuk *authentication signature*s

b. SSD_B (64 bit)
   Untuk menghasilkan *key* untuk proses enkripsi voice dan *signaling messages*.

SSD dapat di-*share* dengan *roaming service providers* sehingga dapat melakukan otentikasi lokal. Sebuah SSD dihasilkan pada saat sebuah mobile kembali pada jaringan semula (*home network*) atau *roam* pada sistem yang lain.

a. Authentication
   Jaringan pada sistem CDMA, mobile menggunakan SSD_A broadcast RAND* sebagai input algoritma CAVE untuk menghasilkan sebuah 18-bit tanda otentikasi (AUTH_SIGNATURE) dan mengirimkan kembali bit ini pada *base station*.

   Signature ini kemudian digunakan oleh *base station* untuk melakukan verifikasi bahwa subscriber valid. Pada dua prosedur, yaitu Global Challenge (dimana semua mobile *challenged* dengan angka random yang sama) dan Unique Challenge (dimana *specific* RAND digunakan untuk setiap *mobile request*) keduanya *available* melakukan otentikasi pada operator.

   Metode Global Challenge dapat melakukan otentikasi dengan sangat cepat selain itu juga pada tracking mobile dan network Call History Count (a 6-bit counter).

b. Voice, Signaling, and Data Privacy
   Mobile phone menggunakan SSD_B dan algoritma CAVE untuk menghasilkan sebuah Private Long Code Mask (diturunkan dari *intermediate value* yang biasa disebut dengan Voice Privacy Mask, yang digunakan di sistem TDMA legal).

   Private Long Code Mask merupakan sebuah algortima, yaitu kunci, key (64 bits), yang dihasilkan oleh Cellular Message Encryption Algorithm (CMEA), dan sebuah Data Key (32 bits).

   Private Long Code Mask digunakan pada mobile dan pada jaringan untuk mengubah karakteristik dari sebuah *Long code*. *Long code* dimodifikasi karena akan digunakan untuk *voice scrambling*, yang membutuhkan tambahan level privacy extra pada *air interface* sistem CDMA.

   Private Long Code Mask tidak mengenkripsikan informasi, hanya saja mengganti *well-known value* yang digunakan pada proses encoding sinyal CDMA dengan sebuah *private value known.*

Perubahan dilakukan di mobile dan juga di network (jaringan). Hal ini dilakukan agar orang-orang yang tidak bertanggung jawab menjadi sangat kesulitan jika ingin mendengarkan secara diam-diam percakapan antara 2 user tanpa mengetahui *Private Long Code Mask*.

Mobile dan network menggunakan CMEA *key* dengan algortima Enhanced CMEA (ECMEA) untuk mengenkripsi *signaling messages* yang dikirimkan ke udara dan digunakan untuk mendeskripsikan informasi yang diterima.

Data key yang terpisah dan algoritma enkripsi yang disebut dengan ORYX digunakan oleh mobile dan network untuk melakukan enkripsi dan deskripsi data traffic pada channel sistem CDMA

Pada design telepon CDMA menggunakan sebuah kode PN (Pseudo-random Noise) yang unik yang dapat digunakan untuk menyebar sinyal sehingga membuat sinyal menjadi sulit diitesepsi.

c. Anonymity
Sistem CDMA mendukung assignment dari Temporary Mobile Station Identifier (TMSI) menjadi sebuah mobile yang dapat digunakan untuk berkomunikasi ke/dari satu mobile tertentu ke mobile yang lain melalui tansmisi udara.

Bagian ini mengakibatkan semakin sulitnya menghubungkan korelasi antara transmisi user mobile dengan user mobile.

3.2.3 CDMA2000 Encryption

Sistem CDMA2000 menggunakan 3 jenis algoritma enkripsi yang berbeda, yaitu: Cellular Authentication and Voice Encryption (CAVE), Cellular Message Encryption Algorithm (CMEA-for signaling), ORYX (for data) on the CDMA channel.

Semua algoritma tersebut berbasiskan stream cipher dengan LFSR. Private Authentication Key (A-key) dan mobile handset Electronic Serial Number (ESN) digunakan sebagai input pada algoritma tersebut baik untuk mobile handset maupun network (HLR/AC).

Input tesebut digunakan sebagai tanda untuk otentikasi dan proses enkripsi/deskripsi voice, data dan signaling messages. Akan tetapi tidak ada satupun dari ketiga cipher yang aman, karena ketiganya sudah berhasil di pecahkan.

4. Algoritma ORYX

4.1 Deskripsi Algoritma

ORYX cipher merupakan standart yang digunakan oleh TIA untuk mengenkripsi data digital yang wireless. Algoritma ini berdasarkan pada Linear Feedback Shift Register. ORYX cipher merupakan stream cipher yang menggunakan keystream generator. Hasil output dari generator ini adalah byte aliran yang pseudo-random. Byte aliran ini kemudian diXORkan dengan plaintext untuk menghasilkan ciphertext dan untuk proses deskripsi dilakukan hal yang sama yaitu mengXORkan ciphertext dengan byte stream yang pseudo-random yang sama.

ORYX merupakan stream cipher sederhana yang berbasis pada *binary linear feedback shift registers* (LFSRs)[1]. Algoritma ini digunakan untuk melindungi data telekomunikasi seluler selama proses transmisi. Cipher ORYX digunakan sebagai *keystream generator* yang menghasilkan byte-byte random. Output dari *keystream generator* ini adalah sekuens *random-looking* dari byte.

Proses enkripsi dapat dilakukan dengan melakukan operasi XOR antara byte-byte keystream yang dihasilkan dengan byte-byte cipherteks. Sedangkan deskripsi dilakukan dengan melakukan operasi XOR antara byte-byte keystream dengan ciphertext. Dengan mengetahui ciphertext dan plaintext maka segmen *keystream* juga dapat diketahui.

Dengan mengetahui pasangan cipherteks dan plainteks yang berkoresponden dapat digunakan untuk mengetahui *keystream* yang berkoresponden pula. Pada makalah ini keamanan algoritma ORYX diperiksa dengan asumsi bahwa kriptanalis menetahui struktur lengkap dari cipherteks dan kunci utama dalam serangan ini hanya berada pada *initial states* dari komponen LFSR.

ORYX cipher terdiri dari 4 komponen utama yaitu:

a. 32 bit Linear Feedback Shift Register dan sebuah S-Box (Wagner).
b. Tiga buah Linear Feedback Shift Register dilambangkan sebagai:
   i. $LFSR_A$
   ii. $LFSR_B$
   iii. $LFSR_C$

S-Box terdiri dari nilai-nilai permutasi (L) dari angka dengan range 0-255.

S-Box merupakan matriks yang berisi subtitusi sederhana yang memetakan satu atau lebih bit dengan satu atau lebih bit lain.

Fungsi Linear Feedback yang digunakan untuk $LFSR_K$ adalah sebagai berikut:

P_K: $x_{n+1} = x_n^{32} + x_n^{28} + x_n^{19} + x_n^{18} + x_n^{16} + x_n^{14} + x_n^{11} + x_n^{10} + x_n^{9} + x_n^{6} + x_n^{5} + x_n + 1$

$LFSR_A$ menggunakan dua buah fungsi Feed Back, yaitu:

P_{A1}: $x_{n+1} = x_n^{32} + x_n^{26} + x_n^{23} + x_n^{22} + x_n^{16} + x_n^{12} + x_n^{11} + x_n^{10} + x_n^{8} + x_n^{7} + x_n^{5} + x_n^{4} + x_n^{2} + x_n + 1$

P_{A2}: $x_{n+1} = x_n^{32} + x_n^{27} + x_n^{26} + x_n^{25} + x_n^{24} + x_n^{23} + x_n^{22} + x_n^{17} + x_n^{13} + x_n^{11} + x_n^{10} + x_n^{9} + x_n^{8} + x_n^{7} + x_n^{2} + x_n + 1$

Dan Shift Register $LFSR_B$ menggunakan fungsi di bawah ini:

P_B: $x_{n+1} = x_n^{32} + x_n^{28} + x_n^{19} + x_n^{18} + x_n^{16} + x_n^{14} + x_n^{11} + x_n^{10} + x_n^{9} + x_n^{6} + x_n^{5} + x_n + 1$
(Wagner).

Permutasi L tidak pernah diganti selama terjadi call. Permutasi ini dihasilkan oleh sebuah algoritma yang menginisialisasi sebuah nilai yang ditransmisikan pada saat setup call pada saat jaringan masih kosong (Wagner).

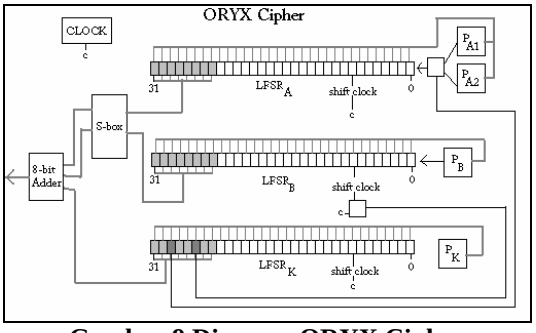
Setiap byte keystream dapat digenerate dengan berbagai cara sebagai berikut:

1) $LFSR_K$ dilakukan satu kali yaitu dengan fungsi P_K

2) $LFSR_A$ dilakukan satu kali yaitu menggunakan salah satu fungsi P_{A1} atau P_{A2}. Pemilihan fungsi tergantung pada isi $LFSR_K$ dari Fungsi dipilih berdasarkan 1 dari 8 bit tertinggi pada $LFSR_K$.

3) $LFSR_B$ dilakukan sekali maupun dua kali. $LFSR_B$ didapatkan berdasarkan 1 dari 8 bit tertinggi yang lain pada $LFSR_K$.

4) Byte yang besar dalam $LFSR_K$ ditambahkan ke byte pada $LFSR_A$ setelah mengalami dipermutasikan oleh L dan byte pada $LFSR_B$ setelah dipermutasikan dengan L kemudian dimodulus 256 untuk menghasilkan 1 byte keystream.
   Misal:
   Keystream = ($High8_K$ + $L[High8_A]$ + $L[High8_B]$) mod 256



Gambar 9 Diagram ORYX Cipher

4.2 Prosedur Serangan terhadap ORYX

Pada serangan ini diasumsikan bahwa kriptanalis mengetahui struktur lengkap dari cipherteks termasuk fungsi *LFSR feedback*. Kuncinya adalah *initial states* dari ketiga 32 bit LFSR, sehingga total panjang key adalah 96 bit. Dengan adanya *key schedule* yang dibuat untuk mempersulit, hal ini dapat mengurangi jumlah *keyspace* sehingga dapat dengan mudah dicari menggunakan *brute force*. Ukuran key berkurang sebanyak 32 bit. Akan tetapi ORYX menjadi algoritma yang tangguh jika menggunakan *key schedule* yang lebih baik dengan 96 bit entropi. Pada serangan yang dibahas dalam makalah ini tidak menggunakan *key schedule* akan tetapi asumsi ini dapat digunakan pada algoritma ORYX yang menggunakan *key schedule*.

Seperti yang sudah dijelaskan di atas, algoritma ORYX mempunyai 96 bit *keyspace*, hal ini sangat tidak mungkin jika hanya dengan menebak semua *initial state* dari *keystream generator* dan mengecek apakan tebakan itu benar atau tidak. Akan tetapi jika *initial state* dari *keystream generator* dapat dibagi menjadi bagian yang lebih kecil dan dimungkinkan untuk menebak satu bagian pada *initial state generator* dan secara inkremental mengecek apakah terkaan tersebut benar sehingga *generator* dapat dipecahkan. Serangan pada makalah ini menggunakan *divide and conquer* dan metode lain yang ada di [9]. Bagian pada ORYX yang dapat digunakan untuk proses serangan ini adalah $LFSR_K$ yang isinya dapat mengkontrol *feedback polynomial* yang dipilih pada $LFSR_A$ dan jumlah step yang diterapkan pada $LFSR_B$ yang ditentukan oleh $LFSR_K$. Keystream dibentuk dari 8 bit dari setiap status dari 3 LFSR, oleh karena itu keyspace dapat dibagi menjadi lebih kecil sehingga serangan difokuskan pada 24 bit tersebut.

4.2.1 Algoritma Serangan

Ditunjukkan bahwa 8 *high bits* dari ketiga LFSR pada satu satuan waktu i^{th} byte *keystream* adalah dihasilkan oleh High8A(i), High8B(i) dan High8K(i).

Initial contents adalah High8A(0), High8B(0) dan High8K(0), serta semua register yang *stepped* sebelum byte pertama pada *keystream*, yaitu Z(1).

Untuk menghasilkan byte *keystream* Z(i + 1) pada instan waktu i + 1, LFSR*K* stepped satu kali, kemudian $LFSR_A$ juga stepped satu kali, kemudian $LFSR_B$ stepped sekali atau dua kali. *Content* dari High8A(i + 1), High8B(i + 1) dan High8K(i + 1) kemudian dikombinasikan untuk membentuk byte *keystream* Z(i + 1). Oleh karena itu, tidak perlu dilakukan terkaan terhadap semua bit (24 bit). Jika dilakukan terkaan terhadap *content* dapat digunakan byte pertama dari *known keystream* Z(1) dan fungsi kombinasi untuk menghitung korspondensi antara *contents* High8K(1). Jadi serangan membutuhkan exhaustive search hanya dengan 16 bit subkey yang merupakan *content* dari High8A(1) dan High8B(1).

Untuk setiap bagian pada 16-bit terkaan terhadap High8A(1) dan High8B(1), digunakan Z(1) dan kemudian dilakukan perhitungan mengenai korespondensi antar *content* High8K(1).

Setelah dilakukan perhitungan serangan dilakukan secara iteratif yaitu untuk menyusun path dari terkaan(prediksi) dari High8A(i), High8B(i) dan High8K(i) yang konsisten dengan *known keystream*. Di setiap iterasi himpunan prediksi untuk keystream berikutnya dibentuk dan hassil terkaan (prediksi) dievaluasi dengan membandingkan byte *known keystream* dengan nilai prediksi.

Pada iterasi ke-i, setelah *stepping* ketiga LFSR untuk menghasilkan output byte berikutnya, High8K(i + 1) dan High8A(i + 1) dapat diketahui bahwa terdapat 1 bit *unknown shifting* sedangkan High8K(i + 1), High8B(i + 1) terdapat 1 atau 2 *unknown* bit untuk setiap byte output.

Dengan mencoba semua kemungkinan kombinasi yang mungkin dari input bit yang baru ini yaitu sebanyak 12 kombinasi, kemudian dilakukan perhitungan terhadap byte hasil output untuk setiap kasus. Maka akan ada 12 byte output yang berbeda yang konsisten dengan terkaan (prediksi) terhadap High8A(i), High8B(i) dan High8K(i). kemudian dilakukan perbandingan antara byte *known keystream* Z(i+1) dengan byte output yang diprediksikan.

Jika Z(i + 1) sama dengan salah satu byte yang diprediksikan untuk kasus dimana terdapat 12 byte output yang saling berbeda maka kemungkinan himpunan nilai tersebut ada pada High8K(i + 1), High8A(i + 1) dan High8B(i + 1). Kemudian nilai tersebut digunakan untuk iterasi berikutnya.

Adakalanya jika terdapat output dengan perbedaan <12 dan byte *keystream* sama dengan byte output hasil prediksi dari lebih dari 1 kombinasi dari input bit baru oleh karena itu dapat disimpulkan bahwa terdapat lebih dari satu kemungkinan himpunan nilai tersebut ada di dalam High8K(i+1), High8A(i+1), dan High8B(i+1). Dengan ini, *path* dari terkaan (prediksi) yang konsisten mungkin akan bercabang, oelh karena itu dibutuhkan *depth- rst search*.

Jika byte *keystream* tidak sama dengan semua byte output yang diprediksikan maka terkaan terdiri dari High8A(i) dan High8B(i) yang jelas tidak benar. Jika hal ini terjadi maka sebaiknya dilakukan backtraking menuju percabangan sebelumnya dan memulai tracking lagi pada path yang lainnya. Jika dilakukan searching pada tiap *path* yang mungkin tanpa mencari path yang

konsisten melakukan prediksi terhadap panjan yang sama dengan jumlah byte dari *known keystream*, maka terkaan terdiri dari High8A(1) and High8B(1) yang jelas salah dan prosedur harus dilakukan ulang untuk membentuk 16-bit yang baru.

Jika ditemukan path panjang yang konsisten diasumsikan bahwa nilai untuk High8A(1), High8B(1) dan High8K(1) adalah benar. Dengan ini dapat memberikan pengetahuan *contents* dari *high eight stages* untuk setiap LFSR dari 3 LFSR pada waktu byte *keystream* pertama kali dihasilkan.

Untuk 24 kali terkaan (prediksi) secara berurutan untuk High8A(i), High8B(i) dan High8K(i) dengan $2 \leq i \leq 25$, dan untuk setiap himpunan nilai untuk High8K(i) dan High8A(i) diberikan bit lain pada *state* $LFSR_K$ dan $LFSR_A$ secara berturu-turut. Sedangkan High8B(i) diberikan nilai dari 1 atau 2 bit pada *state* $LFSR_B$.

Pertama dilakukan rekonstruksi 32-bit state untuk setiap LFSR yaitu pada waktu pertama kali byte *keystream* dihasilkan. LFSR *state* dapat stepped kembali untuk memecahkan *initial states* dari ketiga LFSR yaitu kunci rahasia dari ORYX *generator*.

Jadi dapat dikatakan seluruh kunci dapat dipecahkan menggunakan minimum 25 byte *keystream* dengan prediksi sebanyak 2^{16} terkaan. I*nitial states* tersebut digunakan untuk menghasilak kandidat *keystream* dan kemudian dibandingkan dengan *known keystream*. Jika kandidat *keystream* sama dengan *known keystream*, seranganan berkahir namun jika tidak maka harus dihasilkan 16 bit terkaan baru dan mengulangi prosedur.

Pada prakteknya dimungkinkan dibutuhkan lebih dari 25 byte *keystream* untuk mengatasi ambiguitas pada hasil bit final pada LFSR *states*. Oleh karena itu, dibutuhkan ketelitian lebih lanjut mengenai kemungkin false trails akan terjadi untuk memastikan bahwa bit yang dihasilkan adalah bit yang benar.

4.2.2 Prosedur testing

Performansi dari serangan ini dianalisa berdasarkan seberapa besar kesuksesan yang dapat dilakukan dalam merekonstruksi *initial states* untuk beberapa panjang *keystream*.

Percobaan dilakukan prosedur bahwa *Nonzero initial states* dihasilkan untuk $LFSR_A$;$LFSR_B$ dan $LFSR_K$. *Keystream* dengan panjang segment N, $\{Z(i)\}_{i=1}^{N}$ = 1 dihasilkan menggunakan cipher ORYX.

Outline dari testing procedure adalah:

a. Input: panjang dari *keystream sequence*, N.

b. Initialisation: i = 1, dimana i adalah *current attack index*, dan menetapkan i_{max}, yaitu jumlah maksimal dilakukannya serangan. $LFSR_A$ *initial state seed index*, j merupakan *current* $LFSR_B$ *initial state seed index* dan k merupakan current $LFSR_K$ *initial state seed index*.

c. Stopping Criterion: prosedur testing akan berhenti jika jumlah serangan sudah mencapai i_{max}.

d. Step 1: men-*generate pseudo random initial state seeds* $ASEED_i$, $BSEED_i$ and $KSEED_i$ untuk $LFSR_A$, $LFSR_B$ dan $LFSR_K$, berturut-turut. (pseudorandom number routine drand48[10]).

e. Step 2: men-*generate pseudorandom* LFSR *initial states* menggunakan $ASEED_i$, $BSEED_i$ and $KSEED_i$ untuk $LFSR_A$, LFSRB and $LFSR_K$ secara berturut-turut.

f. Step 3: men-*generate keystream sequence* dari bytes $\{Z(i)\}_{i=1}^{N}$.

g. Step 4: melakukan serangan pada $\{Z(i)\}_{i=1}^{N}$ =1 untuk mendapatkan rekonstruksi dari *initial states* dari 3 LFSR.

h. Step 5: jika $i \leq i_{max}$, *increment* i dan kembali ke step 1

i. Step 6: Stop procedure.

j. Output: rekonstruksi *initial states* dari $LFSR_A$, $LFSR_B$ and $LFSR_K$.

4.2.3 Implementasi Persoalan untuk Serangan

Prosedur serangan yang sudah dijelaskan sebelumnya menggunakan asumsi bahwa terkaan pada beberapa bagian pada High8A(i), High8B(i) dan High8K(i) adalah benar, terkaan ini digunakan lagi untuk memprediksi byte *keystream* selanjutnya, dan digunakan untuk membandingkan byte *keystream* yang diketahui dengan prediksi tersebut.

Jika byte *keystream* bertentangan dengan hasil prediksi maka dapat disimpulakn bahwa terkaan yang digunakan adalah salah. Akan tetapi, sangat mungkin bahwa byte *keystream* Z(i + 1) akan sama dengan 1 atau lebih dari byte hasil prediksi, walaupun nilai untuk High8A(i) dan High8B(i) tidak benar. Hal ini kemudian disebut dengan *false alar*m.

4.2.3.1 Peluang False Alarm

Agar serangan menjadi efeketif maka peluang terjadinya false alarm harus kecil. Oleh karena itu, dibutuhkan peluang yang sangat tinggi untuk pendeteksian terkaan yang salah (melalui perbandingan dengan hasil prediksi yang berkoresponden dengan byte *keystream*). Hal ini berarti dibutuhkan peluang tidak adanya hasil prediksi yang cocok dengan actual keystream secara signifikan harus lebih besar dari 0.5 atau bahwa terkaan terhadap isi High8A(i), High8B(i) dan High8K(i) tidak benar.

Mengingat formasi dari hasil dari prediksi, setiap prediksi dibentuk dari 8-bit nilai yang mungkin untuk High8A(i + 1), 8-bit nilai yang mungkin untuk High8B(i+1) dan 8-bit nilai yang mungkin untuk High8K(i+1). Sehingga jumlah kombinasi dari hasil prediksi tersebut adalah 2^{24}. Hasil prediksi terdiri dari 8 bit. Oleh karena itu dengan sebagian nilai output hasil presiksi dapat dibetnuk multiple kombinasi input yang dapat menghasilkan output tersebut. Jika semua input adalah semua integer non-negatif yang kurang dari 256 dan fungsi kombinasi yang digunakan adalah fungsi modulo 256 dari 3 input, semua nilai output akan sama jika kombinasi terhadap input sama juga. Jadi setiap output dihasilkan dari 2^{16} kombinasi dari input.

Jika 1 dari kombinasi input adalah kombinasi yang benar maka terdapat 2^{16}-1 kombinasi lain, walaupun terdapat kombinasi yang salah, dapat menghasilkan nilai yang sama.

Kemungkinan bahwa satu input kombinasi yang salah dapat menghasilkan output yang sama dengn kombinasi yang benar adalah 2^{16}-1 / 2^{24}-1 ≈ 0.0039. Kmungkinan bahwa satu input kombinasi yang salah menghasilkan sebuah nilai outpu yang berbeda dengan kombinasi yang benar adalah komplemen dari hasil diatas, yaitu 0.9961.

Disini dipilih 12 kombinasi input. Kemungkinan bahwa kombinasi salah dideteksi (dengan membandingkan output dari prediksi yang berkoresponden dengan byte *keystream*) adalah kemungkinan bahwa tidak ada hasil output dari prediksi yang sama dengan nilai *keystream* yang diketahui. Anggap bahwa ke-12 input tersebut adalah tidak benar. Kemudian peluang dapat dihitung dengan distribusi binomial:

$$P(\text{incorrect guess detected}) \approx (0.996)^{12} = 0.9541$$

Karena kemungkinan bahwa tidak ada output hasil prediksi yang cocok dengan byte *keystream*, sehingga terkaan terhadap isi dari High8A(i), High8B(i) dan High8K(i) tidak benar adalah 0:9541, sedangkan kemungkinan bahwa setidaknya 1 output hasil prediksi cocok dengan byte *keystream*, sehingga terkaan terhadap ini dari High8A(i), High8B(i) dan High8K(i) tidak benar adalah 0:0459. Sehingga dapat disimpulkan bahwa kemungkina (peluang) dari *false alarm* adalah kurang dari 5 % dan hal yang penting dalam *false alarm* adalah jika 1 kali berada dalam track yang salah maka terdapat kemungkinan 0:9541track tersebut terdeteksi.

Menggunakan distribusi binomial untuk mengaproksimasi kemungkinan abhwa bit hasil terkaan adalah salah,

$$P(\text{keystream byte matchs 1 prediction}) \approx {}_1C_{12}\,(0.0039)^1\,(0.9961)^{11} = 0.0448$$

$$P(\text{keystream byte matchs 2 prediction}) \approx {}_2C_{12}\,(0.0039)^2\,(0.9961)^{10} = 0.0010$$

P (keystream byte matchs ≥ 2 prediction) ≈ 0.0010

Dengan melihat hasil di atas dapat disimpulkan bahwa dengan bertambahnya waktu akan dihasilkan beberapa kemungkinan kesalahan (1 atau 2). Oleh karena itu lebih baik dilakukan *depth- first search* dari pada mencari pembangkit kesalahan.

Sebagai catatan bahwa jika terdapat *state* yang benar untuk High8A(i), High8B(i) dan High8K(i), tidak pernah dilakukan pencarian terhadap *state* yang salah. Dengan mengidentifikasi s*tate* yang benar pada High8A(1), High8B(1) dan High8K(1) satu kali, dengan cepat dapat ditemukan *state* yang benari untuk for High8A(i), High8B(i) dan High8K(i), untuk $2 \leq i \leq n$ dan $n \geq 25$. Dari sini *initial states* dari 3 LFSR dapat direkonxtruksi.

4.2.3.2 Effect of Length of Known Keystream

Panjang minimum *keystream* yang dibutuhkan agar serangan ini berhasil adalah 25 byte; 1 bit digunakan untuk mendapatkan nilai 8 bit yang dibutuhkan untuk High8K(1), kemudian 8 bit tersebut diberikan kepada setiap 32-bit LFSR yang *initial states*, dan kemudian 1 byte digunakan untuk memecahkan 24 bit lain pada ketiga LFSR *initial states*. Semakin banyak *keystream* yang diberikan, semakin baik proses rekontruksinya. Akan tetapi jika jumlah *keystream* yang diketahui kurang dari 25 byte, serangan masih dapat dilakukan untuk merekontruksi status LFSR states tetapi selanjutnya harus dilakukan exhaustive search pada *content* setiap *stages*.

5. Hasil Percobaan

Serangan berhasil memecahkan *initial states* dengan panjang *keystream* N = 25, 26, 27. untuk setiap panjang *keystream*, serangan dilakukan sampai ribuan kali menggunakan *initial states* LFSR yang *pseudo random*. Serangan berhasil jika *initial state* LFSR yang dihasilkan sama dengan *initial state* LFSR sebenarnya.

N	25	26	27
% sukses	99.7	99.9	100.0

Tabel 2 Succes Rate

6. Serangan dengan Ciphertext-only Attacks

Pada beberapa kasus, serangan menggunakan *known-plaintext attack* dapat diperluas menggunakan *ciphertext-only attack* jika beberapa informasi statistik dari plainteks diasumsikan. Misal, jika plainteks adalah teks dengan bahasa inggris atau data *stereotyped* yang lain, *ciphertext only attack* kemungkinan besar dapat dilakukan hanya dengan ratusan atau ribuan byte cipherteks.

Untuk melakukan *ciphertext only attack*, hal yang pertama kali dilakukan adalah mengidentifikasi kemungkinan string yang mungkin sedikitnya 7 karakter, 1 kata atau frase yang kemungkinan besar muncul dalam plainteks. Misal, kemungkinan string ‘login:_’ dan ‘_ _the _ ’. Kemudian string yang mungkin diselipkan pada setiap posisi cipherteks untuk mendapatkan kemungkinan didapatkannya pesan yang benar.

Jika string plainteks yang benar sudah diketahui maka pengetahuan ini dapat digunakan untuk mendapatkan bagian dari *keystream* dengan panjang yang sama dengan panjang string plainteks yang mungkin.

Seperti yang dijelaskan di atas bahwa *known-plaintext attack* dapat dilakukan pada segmen *keystream* yang telah dihasilkan. Jika setiap terkaan tidak memenuhi sampai akhir dari N=7 byte plainteks maka dapat dikatakan bahwa string tidak cocok untuk posisi tersebut. Sebaliknya maka string tersebut cocok untuk posisi tersebut dengan demikian kemungkinan error dapat dikurangi.

Dengan prosedur di atas diharapkan kesalahan pencocokan menjadi berkurang karena path yang menunjukkan kesalahan telah dieliminasi dengan cepat sebelumnya. Setelah menganalisa byte pertama, maka sisa 2^{16} kemungkinan untuk byte yang besar untuk setiap register.

Pada subbab 3.2.3.1 hanya 0:0459 kemungkinan kesalahan yang tidak terdeteksi setelah dilakukan analisa terhadap byte kedua, dan proporsi setelah byte ketiga adalah 0:0459 dan seterusnya. Hal ini berarti bahwa hanya rata-rata $2^{16} \cdot (0.0459)^6 = 0.00061 \approx 2^{-10.7}$ kemungkinan kesalahan yang tidak terdetaksi untuk mempertahankan test setelah N=7 byte *known plaintext*.

Oleh karena itu dengan jumlah cipherteks < 1000 dapat dikatakan hampir tidak ada kesalahan pencocokan panjang string N=7. Jika digunakan string yang lebih panjang maka dapat mengurangi kemungkinan error.

Pencarian lokasi cipherteks yang menghasilkan *valid match* dengan kemungkinan kata yang digunakan dapat dilakukan dengan cukup efisien. Pencarian ini membutuhkan 2^{16} pengecekan terhadap posisi cipherteks yang mungkin cocok. Dengan < 1000 byte cipherteks usaha yang dibutuhkan mencapai 2^{26}.

Setelah didapatkan kata-kata yang mungkin ada pada plainteks maka sekarang saatnya untuk mencari *key* yang digunakan algoritma ORYX. Setiap *valid match* yang dihasilkan memberikan 8 +(N-1) = 14 bit informasi mengenai *initial states* dari $LFSR_A$ dan $LFSR_K$ dan 8+1.5(N-1) = 17 bit informasi mengenai *initial states* dari $LFSR_B$. Oleh karena itu, dengan 3 kata yang cocok dengan plainteks bisa didapatkan 42 bit informasi untuk setiap 32 bit pada $LFSR_A$ dan $LFSR_K$ dan 51 bit informasi untuk setiap 32 bit pada $LFSR_B$.

Kunci dapat dengan mudah diketahui dengan memecahkan *respective linear equations* dengan GF(2). Sebagai kemungkinan lain, dengan *match* bisa didapatkan 28 bit informasi untuk $LFSR_A$ dan $LFSR_K$, dan 34 bit informasi untuk $LFSR_B$.

Dengan menggunakan exhaustive search terhadap 8 bit sisa yang belum diketahui seharusnya dapat dengan mudah menemukan keseluruhan kunci dengan ± 28 percobaam. Untuk setiap percobaan, dapat dilakukan deskripsi cipherteks dan mengecek apakah hasil deskripsi dapat menghasilkan plainteks yang masuk akal dengan menggunakan analisis statistik frekuensi sederhana atau dengan teknik yang lain.

Dengan mempunyai cipherteks yang cukup dan diketahui beberapa kata yang mungkin ada pada plainteks, seharusnya dapat mempermudah untuk menemukan 2 atau 3 kata yang cocok lainnya sehingga kunci pada ORYX dapat dipecahkan.

Dengan kata lain, serangan dengan *ciphertext-only attacks* pada ORYX kompleksitasnya relatif rendah ketika beberapa pengetahuan plainteks telah diketahui. Komputasi yang dilakukan sangat sederhana dan cipherteks yang dibutuhkan juga tidak banyak sehingga serangan dengan *ciphertext-only attacks* menjadi cukup praktis.

7. Kesimpulan

Kesimpulan yang dapat dimbil dari studi kriptanalis algoritma ORYX adalah:

a. Penggunaan transfer data secara wireless (menggunakan alat telekomunikasi seluler) saat ini semakin berkembang setiap harinya oleh karena itu keamanannya pun perlu ditingkatkan. Walaupun beberapa teknik yang digunakan saat ini tidak unbreakable tetapi teknik tersebut sudah cukup memberikan keamanan yang dibutuhkan. Akan tetapi dengan banyaknya permintaan mengenai keamanan yang lebih akurat maka perlu diadakannya eksplorasi terhadap teknik-teknik baru yang bisa memberikan keamanan yang lebih akurat.

b. ORYX merupakan algoritma *stream cipher* sederhana yang digunakan sebagai *keystream generator* untuk melindungi selama proses transmisi data pada telepon selular khususnya pada sistem CDMA.

c. Teknik *Known Plaintext Attack* yang dibahas di dalam makalah ini memiliki beberapa asumsi bahwa kriptanalis mengetahui struktur cipher secara lengkap dan 96bit kunci rahasia itu hanya merupakan inisial state dari kompunen LFSR.

d. Serangan yang dilakukan terhadap algortima ORYX ini membutuhkan *exhaustive search* lebih dari 16bit dan akan menghasilkan kesuksesan 99% jika kriptanalis mengetahui 25 bit dari *keystream*. Dan kesuksesan semakin meningkat jika kriptanalis dapat mengetahui lebih dari 25 bit dari *keystream*. Terdapat contoh [1], sebuah *keystream* dengan panjang 27 bytes sudah cukup untuk menghasilkan key yang benar untuk setiap percobaan yang dilakukan.

e. Hanya dengan ratusan atau ribuan byte *known ciphertext* ORYX dapat diserang dengan menggunakan metode *Ciphertext only Attack*

f. Dari hasil yang didapatkan dapat diindikasi bahwa algoritma ORYX mempunyai kerentanan keamanan yang sangat rendah.

g. Hampir semua algoritma enkripsi yang digunakan pada telekomunikasi mobile generasi kedua tingkat keamanannya rendah.

DAFTAR PUSTAKA

[1] Cryptanalysis of ORYX. http://.../paper-oryx.pdf.

[2] Cryptanalysis of the Cellular Message Encryption Algorithm. http://.../paper-cmea.pdf

[3] CDMA 1XRTT SECURITY OVERVIEW. http://telecom.co.nz/binarys/ cdma_security_overview.pdf

[4] http://www.isaac.cs. berkeley.edu/isaac/gsm.htm

[5] Mobile Communication Security. http://.../bart_mobile06_gsm_umtsv3.pdf

[6] Munir, Rinaldi. (2006). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[7] E. Dawson and L. Nielsen. Automated cryptanalysis of XOR plaintext strings. Cryptologia, volume XX Number 2, pages 165{181. April 1996.

[8] D. Wagner, B. Schneier and J. Kelsey. Cryptanalysis of the cellular message encryption algorithm. Advances in Cryptology - CRYPTO'97, volume 1294 of Lecture Notes in Computer Science, pages 526{537. Springer-Verlag, 1997.

[9] D. Wagner, B. Schneier and J.Kelsey. Cryptanalysis of ORYX. Unpublished manuscript, 4 May 1997.

[10] H. Schildt. C the Complete Reference Osborne McGraw-Hill, Berkeley, CA, 1990.

[11] Wagner, D. Cryptanalysis of ORYX

[12] Brookson, Charles. http://www.zeata.demon.co.uk/gsmdoc.doc.

[13] Authentication and Security in Mobile Phones by Greg Rose, Qualcomm Inc., Australia.