Studi Perbandingan ORYX Cipher dengan Stream Cipher Standard

Kevin Chandra Irwanto – 13508063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
synolyn@yahoo.com

ABSTRAK

Hampir setiap harinya, kita tidak lepas dari jaringan seluler. Namun, ada kalanya jalur komunikasi itu disadap oleh pihak yang tidak bersangkutan untuk suatu tujuan yang tidak baik. Untuk mencegah hal-hal tersebut maka dipakai suatu algoritma enkripsi untuk penyembunyian isi pesan yang dikirim lewat jaringan seluler.

***Stream cipher* adalah suatu cipher simetris berbasis bit yang beroperasi pada bit tunggal. *Stream cipher* ini sering digunakan untuk penyembunyian isi pesan pada jaringan seluler karena kecepatan proses enkripsi dan dekripsinya dibanding *block cipher*(cipher yang lebih kompleks).**

ORYX merupakan salah satu bentuk dari *stream cipher*. Cipher ini memiliki kompleksitas yang lebih tinggi dibandingkan dengan *stream cipher standard* karena *keyspace*-nya yang cukup panjang. ORYX ini pernah digunakan sebagai algoritma enkripsi untuk penyembunyian transmisi data pada jaringan seluler di Amerika Utara.

Pada makalah ini akan dibahas tentang algoritma ORYX dan perbedaannya dengan *stream cipher standard*. Ada serangan terhadap algoritma ORYX yang dirancang oleh Wagner dkk namun memerlukan kurang lebih 24 *bytes* dari *plaintext* dan 2^{16} buah inisialisasi.

Kata kunci : *Stream Cipher, Cellular System*, ORYX, *keystream*, *security*, LFSR, PRNG,

1. PENDAHULUAN

Pada jaman sekarang, kita sudah tidak asing lagi mendengar kata seluler, salah satu contohnya adalah penggunaan telepon genggam sebagai alat komunikasi. Dalam penyampaian datanya, telepon genggam menggunakan jaringan seluler sebagai media transmisi datanya. Karena jaringan seluler tersebut rawan untuk disadap, maka untuk memproteksi dan mencegahnya beberapa algoritma kriptografi digunakan sebagai sekuritas dari saluran komunikasi tersebut.

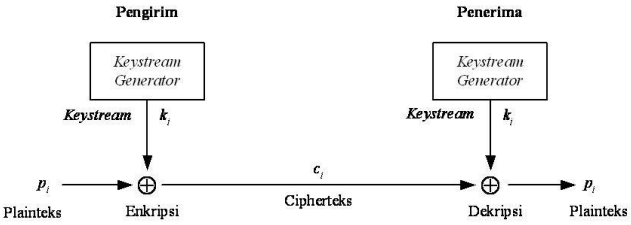
Algoritma yang dipakai adalah *stream cipher*, yang lebih sering digunakan pada aplikasi ini dibandingkan dengan algoritma *block cipher*. Karena kelebihannya dalam proses enkripsi dan juga memiliki kompleksitas hardware yang lebih rendah dibandingkan dengan *block cipher*.

Sedikit informasi tentang *block cipher*, *block cipher* adalah evolusi dari *stream cipher* yang dirasa kurang aman dalam mengenkripsikan suatu data atau pesan. Pada dasarnya, *block cipher* sama dengan *stream cipher* hanya pada proses enkripsinya melibatkan sekaligus banyak bit(1 *block*) dalam satu kali proses enkripsinya. Contoh algoritma *block cipher* antara lain Electronic Code Book(ECB), Cipher Block Chaining(CBC), Cipher Feedback(CFB), dan Output Feedback(OFB).

ORYX merupakan salah satu *stream cipher* yang digunakan untuk aplikasi sekuritas jaringan seluler. Cipher ini mempunyai 96-bit *keyspace*(ruang kunci) yang menyebabkan algoritma ini sulit dipecahkan. Terbukti dengan serangan yang dikemukakan oleh Wagner dkk yang memerlukan inisialisasi awal proses dekripsi sebanyak 2^{16} buah. ORYX pernah digunakan oleh North American Cellular Network(NACN) sebagai metode enkripsi dekripsi data yang dikirim lewat jaringannya.

2. STREAM CIPHER

Stream Cipher adalah cipher simetris berbasis bit yang beroperasi pada bit tunggal. Cipher ini mengenkripsi plainteks menjadi cipherteks bit per bit atau *byte* per *byte* dengan kunci *keystream*. *Stream Cipher* atau cipher aliran ini pertama kali diperkenalkan oleh Vernam melalui algoritmanya, Vernam Cipher. Vernam Cipher diadopsi dari *one-time-pad* yang dalam hal ini, karakter diganti dengan bit atau *byte*.



Gambar 1. Konsep *Stream Cipher*

Baik enkripsi maupun dekripsi menggunakan operator XOR. Pada enkripsi, setiap bit plainteks di-XOR-kan dengan setiap bit *keystream*, sedangkan pada proses dekripsi, setiap bit cipherteks di-XOR-kan dengan setiap bit *keystream* untuk mendapatkan plainteksnya kembali.

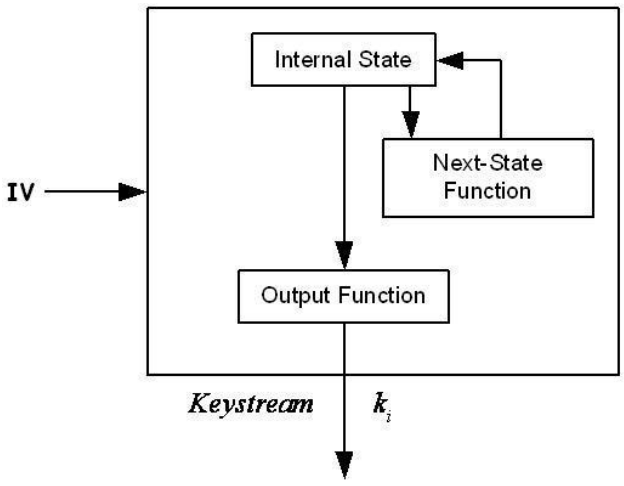
Cipher aliran cocok untuk mengenkripsikan aliran data yang terus menerus melalui sebuah saluran komunikasi, seperti mengenkripsikan data pada saluran yang menghubungkan antara dua atau lebih komputer, juga mengenkripsikan suara pada jaringan telepon *mobile* GSM.

2.1. Keystream (Aliran Kunci)

Dalam cipher aliran, *keystream* atau aliran kunci ini memegang peranan penting dalam tingkat keamanan cipher aliran ini. Suatu *keystream* terdiri atas rangkaian bit yang di-*generate* oleh suatu generator. Salah satu generator yang digunakan cipher ini adalah *Pseudo-Random-Number-Generator*(PRNG).

2.2. Pseudo-Random-Number-Generator

Pseudo-Random-Number-Generator adalah suatu pembangkit bilangan acak yang diinisialisasikan oleh suatu bilangan juga. Inisialisasi bilangan itu disebut dengan *Initialization Vector*(IV). IV ini yang nantinya akan dipakai sebagai 'kunci' dalam suatu proses enkripsi sekaligus dekripsinya.

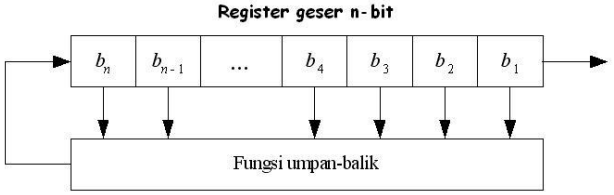


Gambar 2. Proses pada PRNG

Perbedaan antara PRNG dengan pembangkit bilangan acak standard ialah PRNG dapat meng-*generate* bilangan acak yang seluruhnya sama dengan menggunakan IV yang sama. Karena itu PRNG sering juga dipakai dalam metode algoritma kriptografi lainnya.

2.3. Linear Feedback Shift Register(LFSR)

Linear Feedback Shift Register(LFSR) adalah sebuah *keystream generator standard* yang banyak digunakan dalam meng-*generate* kunci-kunci pada algoritma enkripsi. LFSR ini terdiri atas 2 bagian yaitu *register* geser n-bit dan *feedback function*(fungsi umpan-balik). Contoh fungsi umpan-balik yang digunakan adalah dengan meng-XOR-kan bit pertama b_n dengan bit terakhir b_1.



Gambar 3. Konsep Dasar LFSR

Sama halnya dengan PRNG, LFSR memerlukan IV untuk dapat meng-*generate* rangkaian bit kuncinya.

3. ORYX Cipher

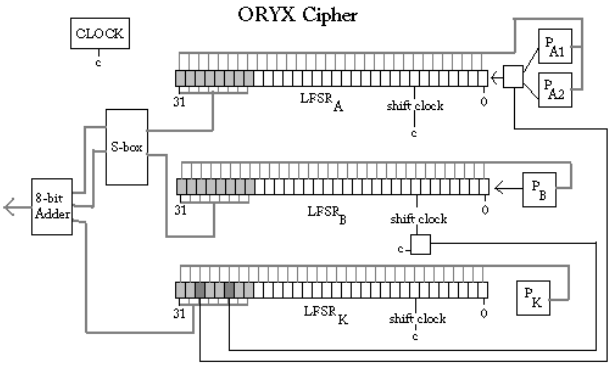
ORYX merupakan salah satu cipher aliran yang berbasis LFSR *standard*. Algoritma enkripsi ORYX ini pernah digunakan oleh Amerika Utara dalam mengenkripsikan transmisi data yang terus menerus dilewatkan melalui jaringan selulernya.

3.1. Deskripsi ORYX Cipher

ORYX ini terdiri dari 3 komponen 32-bit LFSR dan sebuah *S-Box* yang berisi suatu permutasi bilangan bulat. Cipher ini mempunyai 96 bit keyspace.Namun, cipher ini termasuk mudah untuk dipecahkan dengan metode-metode kriptanalis standard seperti *known-plaintext attack* atau *ciphertext-only attack*.

S-Box adalah sebuah matriks yang berisi substitusi sederhana yang memetakan satu atau lebih bit dengan satu atau lebih bit yang lainnya. *S-Box* merupakan satu-satunya langkah nirlanjar di dalam algoritma, karena operasinya adalah *look-up table*. Masukan dari operasi *look-up table* dijadikan sebagai indeks *S-Box*, dan keluarannya adalah entri dari *S-Box*. *S-Box* yang memetakan m bit masukan menjadi n bit keluaran berukuran m x n.

Secara garis besar, ORYX cipher dapat digambarkan sbb :



Gambar 4. Algoritma ORYX

Tiga buah LFSR yang digunakan direpresentasikan dengan $LFSR_A$, $LFSR_B$, dan $LFSR_K$ dengan IV yang sama atau berbeda. Sedangkan S-Box yang digunakan mengandung permutasi bilangan bulat 0-255.

Ada empat fungsi umpan-balik yang digunakan dalam ORYX, sbb:

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

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

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

$$\mathbf{P_B} : x^{n+1} = x^{32} + x^{31} + x^{21} + x^{20} + x^{16} + x^{15} + x^6 + x^3 + x + 1$$

Fungsi umpan-balik P_K digunakan untuk $LFSR_K$. Fungsi P_{A1} dan P_{A2} untuk $LFSR_A$ yang kondisi penggunaannya akan dijelaskan lebih lanjut. Fungsi P_B untuk $LFSR_B$.

Permutasi yang dilakukan oleh S-Box tidak berubah selama terjadinya pemanggilan. Permutasi ini dihasilkan oleh sebuah algoritma yang diinisalisasikan dengan sebuah nilai yang ditransmisikan ketika awal pemanggilan.

Cara mendapatkan *byte-byte* dari aliran kunci untuk algoritma ORYX ini, sbb :

Langkah 1 :

$LFSR_K$ di-*generate* dengan fungsi P_K.

Langkah 2 :

$LFSR_A$ di-*generate* dengan salah satu fungsi, P_{A1} **atau** P_{A2}. Fungsi mana yang dipakai berdasarkan salah satu dari delapan bit tertinggi dari $LFSR_K$(8 bit pertama atau 1 *byte*).

Langkah 3 :

$LFSR_B$ di-*generate* dengan fungsi P_B sebanyak satu **atau** dua kali tergantung salah satu dari delapan bit tertinggi $LFSR_K$ juga.

Langkah 4 :

Byte aliran kunci didapat dengan menjumlahkan 8 bit pertama pada $LFSR_K$ ditambah dengan permutasi dengan S-Box 8 bit pertama $LFSR_A$ ditambah lagi dengan permutasi 8 bit pertama $LFSR_B$ lalu di-modulus 256.

Bit yang digunakan sebagai penentu pengambilan metode pada $LFSR_A$ dan $LFSR_B$ adalah bit yang berbeda satu dengan yang lainnya.

Secara singkat, rumus *byte* dari aliran kunci dengan permutasi L pada S-Box, dapat dituliskan sbb:

$$\text{Keystream} = (\text{High8}_K + L[\text{High8}_A] + L[\text{High8}_B]) \textbf{ mod } 256$$

Proses enkripsi dilakukan seperti biasa, yaitu dengan meng-XOR-kan bit per bit setiap bit plainteks dengan bit *keystream*-nya. Proses dekripsinya juga dilakukan seperti *stream cipher standard*, meng-XOR-kan bit per bit setiap cipherteks dengan biit *keystream*-nya.

3.2. Serangan terhadap ORYX Cipher

Dikarenakan oleh ORYX menggunakan ruang kunci yang cukup banyak, yaitu 3 buah 32 bit LFSR, maka kita hanya memfokuskan pada 8 bit pertama(1 byte pertama) setiap LFSR saja. Namun pada akhirnya, kita akan tetap mencari keseluruhan bit pada setiap LFSR tersebut.

Yang menjadi kunci utama dalam algoritma ORYX ini adalah *Initialization Vector*(IV) dari tiap-tiap LFSR yang ada. Oleh karena itu, pencarian IV tiap-tiap LFSR dapat dilakukan dengan beberapa langkah.

Langkah 1 :

- Tandai 8 bit pertama setiap LFSR dengan $\text{High8}_K(0)$, $\text{High8}_A(0)$, dan $\text{High8}_B(0)$.
- Tandai *byte* pertama aliran kunci dengan Z(1). Z(1) didapat dengan melewatkan isi dari $\text{High8}_K(0)$, $\text{High8}_A(0)$, dan $\text{High8}_B(0)$ ke dalam fungsi $LFSR_K$ satu kali, $LFSR_A$ satu kali dan $LFSR_B$ satu atau dua kali.
- Didapat nilai dari $\text{High8}_K(1)$, $\text{High8}_A(1)$, dan $\text{High8}_B(1)$.

Langkah 2 :

- Nilai $\text{High8}_A(i)$ dan $\text{High8}_B(i)$ dapat menentukan nilai $\text{High8}_K(i)$, karenanya

serangan dapat diminimalisir menjadi 16 bit dari 96 bit keseluruhan.

- Untuk menebak nilai $High8_A(i)$ dan $High8_B(i)$ dapat digunakan nilai Z(i).

Langkah 3 :

- Melakukan sejumlah iterasi untuk mendapatkan beberapa buah nilai kombinasi dari nilai $High8_K(i)$, $High8_A(i)$, dan $High8_B(i)$, karena kita tidak tahu berapa banyak pergeseran bit yang dilakukan.
- Setelah beberapa iterasi, paling banyak akan didapat 12 buah hasil *byte* keluaran yang berbeda yang sesuai dengan tebakan terhadap nilai-nilai $High8_K(i)$, $High8_A(i)$, dan $High8_B(i)$.
- Cocokkan nilai yang didapat tersebut dengan nilai Z(i+1), jika cocok maka didapat sebuah nilai yang pas untuk dilanjutkan ke iterasi berikutnya.
- Terkadang kita dapat menemui situasi ketika terdapat nilai hasil *byte* yang sama dengan yang sudah pernah diperoleh sebelumnya. Hal tersebut mengakibatkan jalur yang diperoleh akan bercabang. Dalam masalah ini, dapat digunakan metode Depth First Search(DFS).
- Atau kadangkala kita menemui hasil keluaran *byte* tidak sama dengan hasil keluaran yang diprediksi, maka dapat dilakukan *backtracking* terhadap proses penebakan nilai $High8_A(i)$ dan $High8_B(i)$.
- Jika tetap tidak menemui hasil lagi sampai akhir percabangan, berarti pemilihan nilai $High8_A(i)$ dan $High8_B(i)$ sebelumnya adalah salah. Lakukan penggantian pada nilai $High8_A(i)$ dan $High8_B(i)$ lalu ulangi **Langkah 3** dan **4**.

Langkah 4 :

- Setelah sukses mendapatkan nilai-nilai tiap LFSR pada akhir proses, lakukan *backtrack* sampai ke awal untuk mendapatkan nilai IV setiap LFSR.
- Namun, hasil backtrack tersebut dapat berbeda dengan hasil prediksi kita waktu awal mula proses dimulai. Hal tersebut menandakan IV yang didapat salah, maka harus mengulang proses dari **Langkah 1** lagi dengan tebakan 16 bit yang berbeda.
- Jika sudah benar, maka proses selesai dan kunci berhasil ditemukan.
- (*optional*) Namun ada kalanya kita tetap memerlukan hasil dari contoh tebakan lainnya untuk menghilangkan ambiguitas bit-bit akhir dari tiap LFSR yang ada.

Metode serangan terhadap ORYX diatas adalah serangan yang dirancang oleh David Wagner, John Kelsey, dan Bruce Schneier yang terbentuk dalam sebuah tim kriptografi *Counterpane System*.

4. ANALISIS

4.1. Analisis ORYX Cipher

ORYX cipher terbukti tingkat keamanannya tinggi dengan serangan yang dilakukan oleh Wagner dkk yang membutuhkan paling sedikit 24 bit plainteks dan iterasi sebanyak 2^{16} kali.

ORYX cipher sebagai salah satu dari cipher aliran, dapat dikatakan lemah jika diserang dengan menggunakan metode serangan *standard* seperti *known-plaintext* ataupun *ciphertext-only* karena cipher ini tetap menggunakan XOR sebagai metode enkripsi dekripsinya, hanya aliran kuncinya saja yang berubah-ubah terus. Untuk membuat ORYX ini menjadi cukup aman, dapat mengganti fungsi XOR yang sudah ada menjadi suatu fungsi simetris yang lain yang cukup aman.

ORYX cipher ini menggunakan fungsi yang membuat ketergantungan antara 3 buah LFSR yang ada. Pemanggilan nilai $LFSR_A$ dan $LFSR_B$ yang bergantung pada nilai $LFSR_K$. Hal tersebut adalah salah satu konsep yang membuat ORYX cipher ini menjadi sulit untuk diterka.

Selain itu, ORYX cipher juga menggunakan metode S-Box(kotak-S) yang termasuk dalam salah satu unsur dalam prinsip perancangan *block cipher*. Ditambah dengan fungsi penjumlahan yang kemudian di-moduluskan.

Beberapa algoritma lainnya yang termasuk dalam cipher aliran adalah *Software-Optimized-Encryption-Algorithm*(SEAL), A5/1, dan RC4. SEAL didesain oleh Don Coppersmith dari IBM Corp., sehingga algoritma tersebut dipatenkan oleh IBM. Sedangkan 40-bit RC4 dapat dipecahkan dengan *brute force*.

Walaupun memecahkan ORYX termasuk sulit, namun SEAL lebih aman dibandingkan ORYX karena sampai sekarang belum ada yang benar-benar berhasil menerjemahkan seluruh plainteks yang dienkripsikan dengan metode SEAL.

4.2. Analisis Stream Cipher Standard

Stream cipher standard dengan tingkat keamanan yang rendah dapat dengan mudah diserang dengan metode *known-plaintext, ciphertext-only,* maupun *flip-bit*. Walaupun hal tersebut sama dengan kondisi ORYX, namun dalam pengaplikasiannya tetap berbeda. Cipher aliran standar ini lebih mudah ditebak karena kunci yang digunakan biasanya menggunakan suatu pola, kata, kalimat, huruf atau angka. Dan menggunakan *keystream generator standard* yang dapat dengan mudah ditebak hasil dari *generator* tersebut. Sehingga dengan salah satu metode *attack* di atas, ruang lingkup serangan dapat dipersempit untuk cipher aliran standar ini.

4.3. Analisis Perbandingan ORYX dengan Stream Cipher Standard

Dalam hal tingkat keamanannya, ORYX dan cipher aliran standar sama-sama dapat dipecahkan dengan metode enkripsi standar, yaitu *known-plaintext attack* dan *ciphertext-only attack*. Namun dalam pengaplikasiannya, memecahkan ORYX lebih sulit dibandingkan dengan cipher aliran standar karena aliran kuncinya lebih acak dibandingkan aliran kunci pada cipher aliran standar. Namun, jika diukur dengan serangan tingkat tinggi yang melibatkan perhitungan-perhitungan dan penerkaan, tingkat keamanan ORYX jauh berada di atas cipher aliran standar karena ruang kuncinya mencapai 3 kali lipat ruang kuncinya.

Dalam hal kecepatan proses, ORYX lebih memakan waktu atau RAM(jika dihitung dengan komputerisasi) karena proses komputasi yang cukup lama dalam menentukan aliran kuncinya. Sedangkan algoritma cipher aliran standar lebih cepat dan tidak memakan banyak RAM. Dikarenakan aliran kunci yang dibangkitkan menggunakan generator standar yang dapat diakses secara langsung. Seperti dalam *library* java atau c++, terdapat fungsi *Random.NextInt(int i)* dalam pengaplikasian generatornya. Namun kecepatan proses dan penggunaan RAM berbanding terbalik dengan tingkat keamanan tingkat tingginya suatu algoritma kriptografi.

5. KESIMPULAN

- Perbandingan ORYX dan *StreamCipher Standard*

Pembanding	ORYX Cipher	Stream Cipher Standard
Ruang Kunci	3n bit	n bit
Tingkat Keamanan Tingkat Standar	Cukup Rendah	Rendah
Tingkat Keamanan Tingkat Tinggi	Tinggi	Rendah
Kecepatan Proses	Lama	Cepat
Penggunaan RAM	Banyak	Sedikit
Penggunaan Metode	S-Box	-

- Metode enkripsi dan dekripsi dari ORYX Cipher tetap menggunakan metode standar yaitu XOR, hanya berbeda pada cara membangkitkan aliran kuncinya saja.

- Mengekspansi banyaknya ruang kunci dapat meningkatkan tingkat keamanan suatu algoritma kriptografi.

- Membuat ketergantungan diantara setiap bit aliran kunci membuat algoritma kriptografi lebih aman. Contohnya dalam *block cipher*, Cipher Block Chaining(CBC).

- S-Box tidak hanya digunakan dalam *block cipher* saja, tetapi juga dapat digunakan dalam *stream cipher*.

- Serangan yang dilakukan terhadap ORYX oleh Wagner dkk tetap memerlukan iterasi dan *initial state* yang cukup banyak.

- Tingkat keamanan suatu algoritma kriptografi tidak hanya diukur dari bagaimana cara plainteks tersebut dienkripsi, tetapi juga bagaimana cara membangkitkan rangkaian aliran kunci yang seacak mungkin.

- Semakin sulit suatu algoritma kriptografi dibuat, semakin sulit algoritma kriptografi tersebut dipecahkan secara menyeluruh.

- Walaupun sudah ada *block cipher* yang lebih aman dibandingkan dengan *stream cipher*, namun *stream cipher* tetap digunakan karena kecepatan prosesnya yang cepat dan penggunaan RAM-nya yang relatif lebih sedikit dibandingkan *block cipher* yang memakan lebih banyak RAM dan proses yang lama pula.

6. REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF5054*, Departemen Teknik Informatika, Institut Teknologi Bandung, 2005.

[2] Neel, Joshua. *Cryptanalysis of Mobile Phone Cryptology*. (www.cs.umd.edu/Honors/reports/cryptanalysis.doc). Diakses tanggal 22 Maret 2011.

[3] Wagner, David ; J. Kelsey, B. Schneier. *Cryptanalysis of ORYX*. (www.schneier.com/paper-oryx.pdf). Diakses tanggal 22 Maret 2011.

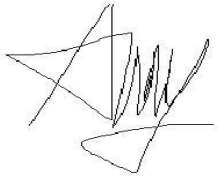
[4] www.kremlinencrypt.com/algorithms.htm Diakses tanggal 22 Maret 2011.

[5] Wagner, David ; J. Kelsey, B. Schneier. *Cryptanalysis of the Cellular Message Encryption Algorithm*. Springer-Verlag, 1997.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 23 Maret 2011



Kevin Chandra Irwanto
13508063