Analisis Perbandingan *Stream Cipher* RC4 dan SEAL

A. Thoriq Abrowi Bastari - 13508025
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18025@students.if.itb.ac.id

***Abstrak*— Pada makalah ini akan dibahas mengenai perbandingan antara 2 *stream cipher* yang cukup umum digunakan, yaitu RC4 dan SEAL (Software-Optimized Encryption Algorithm). *Stream cipher* (*cipher* aliran) adalah salah satu algoritma *cipher* berbasis bit. *Cipher* aliran mengenkripsi *plaintext* bit per bit menjadi *ciphertext*. Dengan kata lain, enkripsi dilakukan sebanyak jumlah bit dari *plaintext* tersebut. Namun, terkadang enkripsi *plaintext* juga dapat dilakukan per byte. Terdapat beberapa algoritma *cipher* yang termasuk dalam kategori ini, di antaranya adalah WAKE, Cellular Automaton, RC4, SEAL, A2, dan A5. Penulis akan mencoba membahas lebih dalam dua dari algoritma-algoritma tersebut, yaitu RC4 dan SEAL. Penulis memilih membandingkan kedua *stream cipher* ini karena akan cukup menarik melakukan perbandingan antara keduanya. RC4 dianggap sebagai algoritma *stream cipher* yang cepat dan *robust*, sementara SEAL diklaim sebagai algoritma *stream cipher* tercepat. Tujuan dilakukan studi perbandingan ini adalah untuk mengetahui kelebihan dan kekurangan dari algoritma *stream cipher* RC4 dan SEAL. Dengan mengetahui hal tersebut diharapkan baik penulis maupun pembaca makalah ini dapat mengenal dengan lebih baik karakteristik dari kedua algoritma ini sehingga bisa memilih dengan tepat apabila dihadapkan pada pilihan yang menyangkut algoritma RC4 dan SEAL.**

***Kata kunci*—*stream cipher*, RC4, SEAL**

I. PENDAHULUAN

Seiring dengan berkembangnya teknologi informasi, informasi sudah bukan barang mahal lagi. Hampir sebagian besar informasi yang diperlukan dapat diperoleh melalui berbagai sumber, seperti internet, dengan menggunakan berbagai jenis perangkat yang mempermudah pencarian informasi, seperti Google.

Kemudahan dalam memperoleh informasi ini ditanggapi dengan sangat positif oleh semua orang. Namun, timbul masalah-masalah baru yang mengiringi hal tersebut. Isu yang paling penting adalah isu keamanan dalam berinteraksi, atau keamanan informasi.

Seseorang yang ingin mengirimkan atau menyimpan informasi yang bersifat *confidential* menjadi ragu tentang keamanan informasi yang dikirimnya tersebut. Pada era teknologi yang sudah maju seperti sekarang ini, hal tersebut dapat dimaklumi. Tidak ada yang dapat menjamin dengan sangat pasti bahwa suatu informasi penting yang dikirimkan dari suatu komputer, telepon genggam, *smart phone*, atau pun alat-alat elektronik lainnya tidak dapat dilihat oleh orang lain yang tidak mempunyai hak.

Isu tersebut lah yang menyebabkan ilmu kriptografi semakin berkembang luas dan umum dalam masyarakat. Kriptografi adalah keilmuan dan praktek dalam menyembunyikan informasi. Sebenarnya kriptografi sendiri bukan ilmu yang baru muncul pada era modern ini. Sudah dari jauh hari orang-orang berusaha menjaga keamanan informasi yang *confidential*. Namun, seperti yang telah disebutkan sebelumnya, kemudahan dalam memperoleh informasi pada era modern ini lah yang menyebabkan kriptografi semakin populer, diperlukan, dan banyak digunakan.

Sudah banyak algoritma yang diciptakan untuk melakukan enkripsi pada data. Sudah banyak juga upaya yang dilakukan untuk mencari celah pada algoritma yang telah diciptakan untuk menguji kekuatannya. Algoritma yang berkembang akhir-akhir ini adalah algoritma kriptografi modern. Berbeda dengan algoritma kriptografi klasik yang beroperasi dalam mode karakter, algoritma kriptografi modern beroperasi dalam mode bit. Hal ini didorong oleh semakin umumnya penggunaan komputer digital baik sebagai media penyimpanan maupun perpindahan informasi.

Algoritma kriptografi modern dapat dikategorikan menjadi dua jenis, yaitu *stream cipher* (*cipher* aliran) dan *block cipher* (*cipher* blok). *Cipher* blok melakukan enkripsi dan dekripsi per blok bit, misalnya 32-bit atau 64-bit. Sementara *cipher* aliran beroperasi pada bit tunggal, berarti enkripsi dan dekripsi dilakukan setiap satu bit.

Algoritma RC4 dan SEAL yang akan dijelaskan oleh penulis merupakan dua dari banyak algoritma *cipher* aliran.

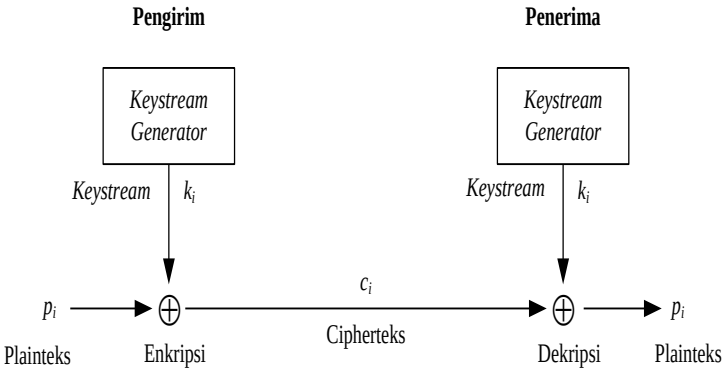
II. TEORI DASAR

"*Block ciphers operate on data with a fixed transformation on large blocks of plaintext data; stream ciphers operate with a time-varying transformation on individual plaintext digits*." (Rainer Ruepper)

Stream cipher adalah salah satu jenis algoritma kriptografi modern yang mengenkripsi plainteks menjadi cipherteks bit per bit atau byte per byte. *Stream cipher*

atau *cipher* aliran adalah sebuah *symmetric key cipher* yang menggunakan *keystream* untuk kemudian digabungkan dengan *plaintext* untuk menghasilkan sebuah *ciphertext*. Operasi yang umum digunakan pada *stream cipher* adalah operasi XOR. Algoritma *stream cipher* pertama kali diperkenalkan oleh Vernam lewat algoritma yang diciptakannya, yaitu Vernam Cipher.

Skema pada *cipher* aliran pada umumnya adalah sebagai berikut:



Gambar 1. Skema *stream cipher*

Dari skema di atas dapat diperoleh informasi bahwa keamanan *stream cipher* sepenuhnya bergantung pada *keystream* yang digunakan. *Keystream* pada *stream cipher* dihasilkan oleh *keystream generator*. Oleh karena itu, *keystream generator* merupakan elemen yang paling penting dalam *cipher* aliran.

Secara umum terdapat tiga kasus *keystream* yang dihasilkan oleh *keystream generator*, yaitu:

1. Seluruhnya 0
2. Berulang secara periodik
3. Sepenuhnya acak

Apabila *keystream* yang dihasilkan seluruhnya adalah 0, maka *cipher* aliran menjadi tidak berguna karena tidak akan ada perubahan yang terjadi, dengan kata lain plainteks sama dengan cipherteks (P XOR 0 = P).

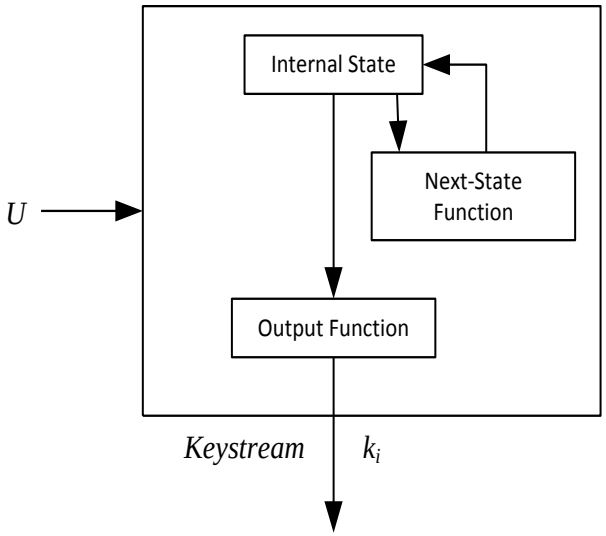
Sementara jika *keystream* yang dihasilkan oleh *keystream generator* berulang secara periodik, maka algoritma enkripsi yang dilakukan akan sama dengan algoritma dengan XOR biasa yang memiliki tingkat keamanan yang rendah.

Tetapi apabila *keystream generator* dapat mengeluarkan *keystream* yang benar-benar acak, maka algoritma yang digunakan memiliki tingkat keamanan yang sempurna. Panjang plainteks sama dengan panjang *keystream* yang sepenuhnya acak, berarti sama dengan *one-time pad*, yang disebut sebagai *unbreakable cipher*.

Permasalahannya, tidak mungkin menghasilkan *keystream* yang benar-benar acak. Hal ini diakali dengan membuat fungsi *pseudorandom* yang rumit dan sulit untuk ditebak. Semakin baik fungsi *pseudorandom* yang digunakan semakin acak *keystream* yang dihasilkan, maka akan semakin sulit pula untuk dipecahkan oleh kriptanalis.

Untuk menghasilkan *keystream*, *keystream generator* memerlukan suatu kunci U. Pada saat membangkitkan *keystream*, *keystream generator* akan menggunakan kunci U yang diperoleh untuk mengacak bit atau byte yang akan dikeluarkan untuk setiap langkahnya sehingga menghasilkan *keystream* berupa *stream of bit/byte* yang acak. Kunci U ini lah yang nantinya harus dimiliki juga oleh penerima untuk membangkitkan *keystream* yang sama dengan *keystream* yang digunakan untuk melakukan enkripsi data.

Untuk lebih jelasnya, proses pembangkitan *keystream* diilustrasikan oleh gambar di bawah.



Gambar 2. Proses dalam *keystream generator stream cipher*

Dalam proses pembangkitan *keystream* dalam *keystream generator* biasanya terdapat sebuah Internal State yang bila digabungkan dengan kunci U dan fungsi keluaran akan menghasilkan bit atau byte yang bersifat *pseudorandom*. Proses ini diulang setiap kali sepanjang plainteks yang akan dienkripsi.

Beberapa kelebihan dari *stream cipher* di antaranya adalah kecepatannya dieksekusi oleh *hardware*, sederhana untuk dibuat, dan kemampuannya untuk mengenkripsi plainteks dengan panjang yang tidak diketahui sebelumnya.

Beberapa *stream cipher* baru dikhususkan untuk implementasi pada *software*, contohnya adalah RC4 dan SEAL. Selain itu, *cipher stream* juga dipakai dalam koneksi nirkabel karena dapat mengenkripsi plainteks dengan panjang yang belum jelas.

A. RC4 Stream Cipher

RC4 adalah *stream cipher* yang diciptakan oleh Ron

Rivest pada tahun 1987. RC sendiri merupakan singkatan dari Rivest Cipher (kadang juga disebut Ron's Code). Pada awalnya algoritma RC4 ini dirahasiakan sampai dikirimkan ke milis *Cypherpunks* secara anonim pada tahun 1994.

Setelah itu algoritma RC4 mulai diketahui banyak orang dan menyebar luas melalui internet. RC4 pun menjadi populer dan banyak dipakai untuk berbagai kegunaan, terutama dalam koneksi internet seperti Wi-Fi. Faktor-faktor dalam kesuksesan RC4 antara lain adalah kecepatan, efisien untuk diimplementasikan baik dalam *hardware* maupun *software*, dan mudah untuk dikembangkan.

Serupa dengan skema umum *stream cipher*, RC4 membangkitkan *keystream* dengan *keystream generator* lalu dilakukan XOR antara *key* tersebut dengan plainteks. Algoritma enkripsi RC4 beroperasi dalam byte, berarti XOR dilakukan setiap satu byte plainteks dengan satu byte *key* dari *keystream*. RC4 menggunakan fungsi dekripsi dan enkripsi yang sama karena operasi yang dilakukan hanyalah XOR antara *keystream* yang dibangkitkan dengan plainteks.

Keystream generator pada RC4 menggunakan Initial State yang berupa S-box yang berukuran 16x16. Pertama S-box diinisialisasi secara linear, berarti S[i]=i. Lalu dilakukan permutasi isi S-box dengan memanfaatkan kunci yang dimiliki.

Algoritma dari RC4 secara umum adalah sebagai berikut:

1. Inisialisasi S-box dengan linear
2. Lakukan *padding* pada kunci apabila panjang kunci kurang dari 256
3. Permutasi isi dari S-box dengan mengacaknya menggunakan kunci
4. Bangkitkan *keystream* lalu di-XOR-kan dengan plainteks

Permutasi pada S-box dilakukan seperti pada *pseudocode* berikut (S adalah S-box dan K adalah kunci yang telah di-*padding* jika perlu):

```
for i = 0 to 255:
j = (j + S[i] + K[i]) mod 256
swap S[i] and S[j]
```

Sementara itu dalam membangkitkan *keystream* algoritma yang dilakukan adalah sebagai berikut:

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap S[i] and S[j]
t = (S[i] + S[j]) mod 256
K = S[t]
```

Setelah itu akan dilakukan operasi XOR antara K yang diperoleh dan satu byte plainteks, P[i]. Operasi permutasi dan pembangkitan *keystream* yang dilakukan sebelumnya sudah cukup baik untuk membuat K sebuah byte yang acak. Bahkan dengan permutasi tersebut bisa terdapat sebanyak 2^{1700} ($256! \times 256^2$) kemungkinan S-box.

B. SEAL Stream Cipher

Selain RC4, SEAL juga merupakan salah satu *stream cipher* yang paling populer dan dianggap paling cepat. Versi pertama dari algoritma enkripsi SEAL dipublikasikan oleh Don Coppersmith dan Phillip Rogaway pada tahun 1994 di IBM. Sementara versi yang terakhir beredar, yaitu versi 3.0, dipublikasikan pada 1997. Algoritma SEAL ini telah dipatenkan atas nama IBM.

SEAL, sesuai dengan namanya, adalah algoritma enkripsi *stream cipher* yang optimal untuk digunakan dalam perangkat lunak. Untuk dapat bekerja dengan maksimal, dibutuhkan prosesor 32-bit untuk menjalankan algoritma ini. *Computational cost* yang dibutuhkan pada prosesor 32-bit adalah sebesar 4 *clock cycles per byte*.

Beberapa ide utama dalam pembuatan *cipher* aliran ini adalah sebagai berikut:

- Menggunakan S-box yang berukuran besar dan bersifat rahasia. Isi dari S-box bersifat *key-driven* atau tergantung dari kunci yang digunakan.
- Bergantian menggunakan operasi aritmatik XOR dan penambahan.
- Menggunakan Internal State yang dikelola oleh *cipher*.
- *Round function* tergantung pada *round number* dan *iteration function* yang tergantung pada *iteration number*.
- Menggunakan metode-metode yang simpel dan populer, seperti SHA-1.

SEAL *cipher* bukan lah *stream cipher* yang biasa. Algoritma enkripsi ini tergolong dalam *pseudorandom function family* atau PRF. SEAL memerlukan tiga parameter yang untuk dieksekusi, yaitu a, n, dan L. SEAL(a, n, L) memetakan 32-bit *string* n kepada string berukuran L-bit dengan a sebagai kuncinya. Sebagai fungsi *pseudorandom* SEAL(a, n, L) harus terlihat seperti fungsi yang acak apabila a acak dan tidak diketahui.

Salah satu karakteristik utama SEAL *cipher* terletak pada S-box yang digunakannya. Proses inisialisasi dari S-box tersebut adalah seperti ini:

procedure Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$

$A \leftarrow n \oplus R[4\ell]$;
$B \leftarrow (n \ggg 8) \oplus R[4\ell + 1]$;
$C \leftarrow (n \ggg 16) \oplus R[4\ell + 2]$;
$D \leftarrow (n \ggg 24) \oplus R[4\ell + 3]$;

for $j \leftarrow 1$ **to** 2 **do**
  $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$;
  $P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \ggg 9$;
  $P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$;
  $P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \ggg 9$;

$(n_1,\ n_2,\ n_3,\ n_4) \leftarrow (D,\ B,\ A,\ C)$;

$P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$;
$P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \ggg 9$;
$P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$;
$P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \ggg 9$;

Gambar 3. Proses inisialisasi S-box pada SEAL *cipher*

Sementara pemetaan dilakukan dengan algoritma seperti berikut:

function SEAL(a, n, L)

$y = \lambda$;
for $\ell \leftarrow 0$ **to** ∞ **do**
  Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$;
  for $i \leftarrow 1$ **to** 64 **do**
1 $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \ggg 9$; $B \leftarrow B \oplus A$;
2 $Q \leftarrow B$ & 0x7fc; $C \leftarrow C \oplus T[Q/4]$; $B \leftarrow B \ggg 9$; $C \leftarrow C + B$;
3 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \ggg 9$; $D \leftarrow D \oplus C$;
4 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A \oplus T[Q/4]$; $D \leftarrow D \ggg 9$; $A \leftarrow A + D$;
5 $P \leftarrow (P + A)$ & 0x7fc; $B \leftarrow B \oplus T[P/4]$; $A \leftarrow A \ggg 9$;
6 $Q \leftarrow (Q + B)$ & 0x7fc; $C \leftarrow C + T[Q/4]$; $B \leftarrow B \ggg 9$;
7 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D \oplus T[P/4]$; $C \leftarrow C \ggg 9$;
8 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A + T[Q/4]$; $D \leftarrow D \ggg 9$;
9 $y \leftarrow y \ \|\ B + S[4i-4] \ \|\ C \oplus S[4i-3] \ \|\ D + S[4i-2] \ \|\ A \oplus S[4i-1]$;
10 **if** $|y| \geq L$ **then return** $(y_0 y_1 \ldots y_{L-1})$;
11 **if** $odd(i)$ **then** $(A,\ B,\ C,\ D) \leftarrow (A + n_1,\ B + n_2,\ C \oplus n_1,\ D \oplus n_2)$
      else $(A,\ B,\ C,\ D) \leftarrow (A + n_3,\ B + n_4,\ C \oplus n_3,\ D \oplus n_4)$;

Gambar 4. Skema umum fungsi SEAL *cipher*

III. PENERAPAN ALGORITMA *STREAM CIPHER*

A. Penjelasan Umum

Untuk lebih memahami *stream cipher* penulis mengimplementasikan algoritma *cipher* aliran RC4 dalam bahasa pemrograman C#. Berikut adalah kode sumber dari program uji yang dibuat:

Algorithm.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    class Algorithm
    {
        private static List<byte>
ciphertext = new
List<byte>();//ciphertext to be sent
        private static byte[] state =
new byte[256];//internal state
        private static byte[]
keystream;//keystream generated by
keystream generator

        private static void
KeystreamGenerator(List<byte> data,
String key)
        {//generate keystream
            keystream = new
byte[data.Count];
            for (int i = 0, j = 0; i <
256; i++)
            {//initializing internal
state
                j = (j + state[i] +
key[i]) % 256;
                byte temp = state[i];
                state[i] = state[j];
                state[j] = temp;
            }
            for (int i = 0, j = 0, idx
= 0; idx < data.Count; idx++)
            {//generate pseudorandom
key
                i = (i + 1) % 256;
                j = (j + state[i]) %
256;
                byte temp = state[i];
                state[i] = state[j];
                state[j] = temp;
                int t = (state[i] +
state[j]) % 256;
                byte k = state[t];
                keystream[idx] =
state[t];

            }
        }

        public static List<byte>
RC4Cipher(List<byte> data, String key)
        {
            List<byte> ciphertext =
new List<byte>();
            for (int i = 0; i < 256;
i++) //S-box initialization
            {
```

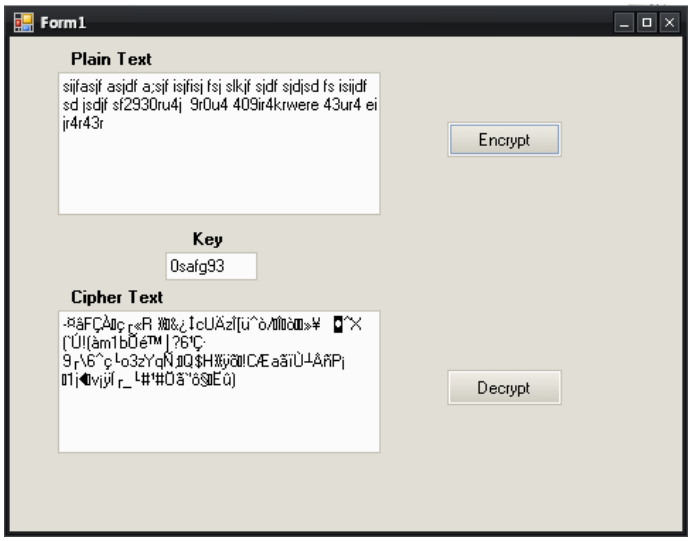
```
                state[i] = (byte)i;
            }
            if (key.Length < 256)
//padding
            {
                StringBuilder sb = new
StringBuilder(key);
                int n = key.Length;
                while (n < 256)
                {

sb.Append(key[n%key.Length]);
                    n++;
                }
                key = sb.ToString();
            }

            KeystreamGenerator(data,
key);//generate keystream

            for (int idx = 0; idx <
data.Count; idx++)
            {//XOR keystream and
plaintext
                int c = keystream[idx]
^ data[idx];

ciphertext.Add((byte)c);
            }
            return ciphertext;
        }
    }
}
```

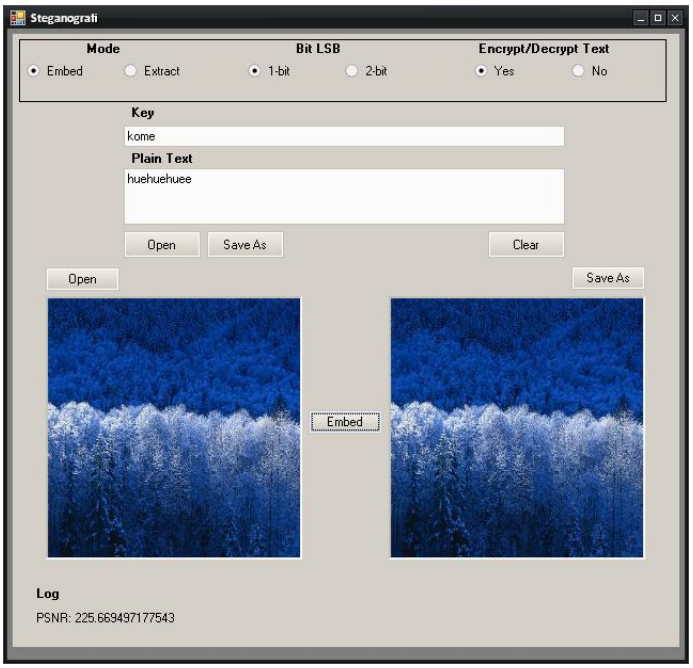
Form1.cs

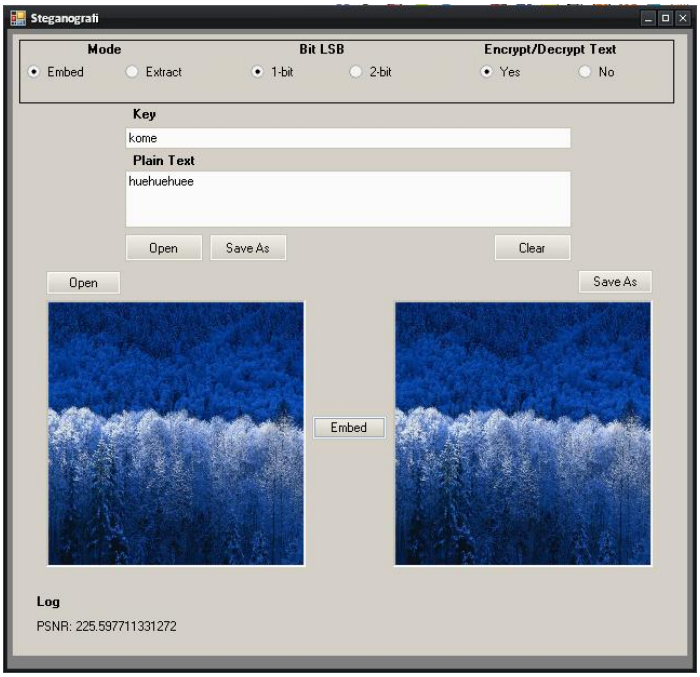
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void
button1_Click(object sender, EventArgs
e)
        {
            String plain =
textBox1.Text;
            //convert into byte stream
            List<byte> byteStream =
new List<byte>();
            for (int i = 0; i <
plain.Length; i++ )
            {

byteStream.Add(Convert.ToByte(plain[i]
));
            }
            //call the cipher method
            List<byte> ciphertext =
Algorithm.RC4Cipher(byteStream,
textBox3.Text);
            //convert back into string
            StringBuilder sb = new
StringBuilder();
            for (int i = 0; i <
ciphertext.Count; i++)
            {

sb.Append(Convert.ToChar(ciphertext[i]
));
            }
            textBox2.Text =
sb.ToString();
        }

        private void
button2_Click(object sender, EventArgs
e)
        {
            String cipher =
textBox2.Text;
            //convert into byte stream
            List<byte> byteStream =
new List<byte>();
            for (int i = 0; i <
cipher.Length; i++)
            {

byteStream.Add(Convert.ToByte(cipher[i
]));
            }
            //call the cipher method
            List<byte> plaintext =
Algorithm.RC4Cipher(byteStream,
textBox3.Text);
            //convert back into string
            StringBuilder sb = new
StringBuilder();
            for (int i = 0; i <
plaintext.Count; i++)
            {

sb.Append(Convert.ToChar(plaintext[i])
);
            }
            textBox1.Text =
sb.ToString();
        }
    }
  }
```

File Algorithm.cs merupakan *file* yang berisi kode algoritma enkripsi *cipher stream* RC4. Pada kode sumber tersebut diimplementasikan 2 fungsi yang digunakan, yaitu fungsi KeystreamGenerator yang mengisi variabel keystream. Sementara itu, fungsi RC4Cipher melakukan inisialisasi Internal State, melakukan *padding* pada kunci key jika diperlukan, lalu memanggil fungsi KeystreamGenerator. Setelah keystream diisi, fungsi ini akan melakukan XOR kepada seluruh byte plainteks dengan keystream yang tadi dihasilkan.

Form1.cs digunakan sebagai antarmuka dalam menggunakan algoritma *cipher* RC4 yang diimplementasikan. Berikut adalah contoh penggunaan dan pengujian program tersebut:







Gambar 5. Pengujian Algoritma RC4

Pengujian pertama dilakukan pada form yang telah dibuat sebelumnya. Sementara pengujian selanjutnya dilakukan dengan memasukkan algoritma RC4 ke dalam steganografi gambar bitmap.

Gambar kedua adalah pengujian steganografi menggunakan algoritma enkripsi RC4. Nilai PSNR yang dihasilkan kira-kira sebesar 225.6695. Untuk perbandingan, pada gambar yang selanjutnya pengujian steganografi digunakan dengan menggunakan Vigenere Cipher. Nilai PSNR yang dihasilkannya adalah kira-kira sebesar 225,5977.

Dari pengujian tersebut didapat bahwa nilai PSNR yang dihasilkan menggunakan algoritma enkripsi RC4 sedikit lebih besar, yang berarti gambar lebih tidak terganggu kualitasnya.

IV. ANALISIS

Dari hasil pengujian dan studi yang dilakukan oleh penulis didapatkan bahwa RC4 adalah algoritma enkripsi *stream cipher* yang sangat cepat dan memiliki tingkat keamanan yang relatif baik. Meskipun kekuatan enkripsinya tidak sebaik *cipher-cipher* lain yang ditemukan setelah RC4 ini, namun kelebihannya dalam hal kecepatan dan efisiensi penggunaannya baik pada *software* maupun *hardware* membuat algoritma ini populer digunakan.

Kelebihan lain yang tidak kalah pentingnya adalah kesederhanaannya. *Programmer* mana pun seharusnya dapat mengimplementasikan algoritma *cipher* aliran RC4 dalam waktu yang relatif singkat. Namun dengan kekuatan *cipher* yang cukup mumpuni.

Hal lain yang diperoleh adalah kemampuannya dalam

enkripsi dalam *file* bitmap yang sedikit lebih baik dari pada *cipher* klasik. Meskipun tujuan utama *stream cipher* ini sebenarnya adalah dalam enkripsi data dalam jaringan komputer, ternyata untuk penggunaan yang lain pun algoritma ini masih dapat memberikan hasil yang baik.

Sementara itu, meski tidak melakukan uji program pada *cipher* aliran SEAL, penulis dapat membandingkannya dengan RC4. SEAL *cipher* dalam proses enkripsinya perlu membangkitkan terlebih dahulu tiga buah tabel(S-box) berukuran besar yang isinya ditentukan oleh kunci yang diterima. Hal tersebut membuat waktu inisialisasi dalam enkripsi plainteks menjadi sedikit lebih lama pada awalnya. Namun, untuk plainteks yang panjang SEAL justru merupakan salah satu algoritma enkripsi yang tercepat, apabila tidak yang paling cepat.

Hal itu juga yang menyebabkan SEAL tidak cocok digunakan apabila memori yang dimiliki tidak cukup besar. Tabel-tabel yang dibangkitkan oleh SEAL membutuhkan 3-4 Kbyte memori.

Kelebihan lain dari SEAL *cipher* adalah tingkat keamanannya yang sangat tinggi. *Cipher* ciptaan Don Coppersmith ini sampai sekarang belum ada yang dapat memecahkan/mempublikasikan kriptanalisisnya.

V. KESIMPULAN

Kesimpulan yang dapat diambil dari makalah ini adalah sebagai berikut:

1. *Stream cipher*, berbeda dari *cipher* blok, dapat melakukan enkripsi dengan sangat cepat dan efisien.
2. RC4 merupakan algoritma *cipher* yang cepat, efisien, dan mudah untuk diimplementasikan.
3. SEAL *cipher* memakan memori yang relatif besar dibandingkan *cipher-cipher* yang lain. Namun dapat dengan sangat cepat melakukan enkripsi setelah inisialisasi tabel selesai dilakukan.

Pengembangan dari algoritma ini bisa dilakukan dengan menambahkan/menggantinya dengan algoritma program dinamis. Namun, dengan begitu kalkulasi yang dilakukan oleh program akan sangat berat dan sangat berpotensi untuk mengurangi kinerja program.

REFERENSI

[1] Phillip Rogaway & Don Coppersmith, "A Software-Optimized Encryption Algorithm,"

[2] Munir. Rinaldi, *Diktat Kuliah IF3058 Kriptografi*, 2006, Program Studi Teknik Informatika Bandung.

[3] http://www.kremlinencrypt.com/algorithms.htm

[4] Schneier, Bruce, "Protocols, Algorithms, dan Source Code in C", John Wiley & Sons,Inc.

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, 22 Maret 2011

A Thoriq Abrowi Bastari (13508025)