ANALISIS PERBANDINGAN DAN PENGUJIAN RIVEST CIPHER 4 DAN CIPHERSABER-2

I.Y.B. Aditya Eka Prabawa W.

Laboratorium Ilmu dan Rekayasa Komputasi, Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganeca 10, Bandung 40132, Indonesia
e-mail: aditya_eka@students.itb.ac.id

ABSTRAK

Makalah ini membahas algoritma kriptografi kunci-simetri modern, yaitu cipher aliran (*stream cipher*) RC4 (Rivest Cipher 4) dan analisis perbandingannya dengan CS2 (CipherSaber-2) yang sejenis. Dalam makalah ini dibahas secara singkat sejarah kedua algoritma, kemudian dibahas juga secara dalam konsep, teori, dan algoritma enkripsi RC4 dan CS2 serta analisis perbandingan antara keduanya.
Untuk mendukung analisis dan pengujian algoritma RC4 dan CS2, telah dibuat sebuah program kecil yang dapat mengenkripsi dan mendekripsi berkas apapun (teks, dokumen Microsoft Office, gambar, audio, video, executable, dan sebagainya) dengan kedua algoritma enkripsi tersebut. (Pada proposal makalah ini, disebutkan bahwa sebagai tambahan akan dibuat juga Windows Sidebar Gadget yang memiliki kemampuan yang sama dalam bentuk gadget untuk Windows Sidebar. Namun karena keterbatasan waktu, Windows Sidebar Gadget terebut belum sempat diselesaikan bersamaan dengan selesainya penulisan makalah ini).
Selain itu juga dibahas kekuatan, kelemahan, dan jenis-jenis serangan (*attack*) pada RC4 dan CS2, serta aplikasi dan penerapan dari enkripsi RC4 dan CS2 pada berbagai sistem keamanan.
RC4 yang dibahas pada makalah ini adalah spesifikasi standar algoritma RC4, yang tidak dimodifikasi dengan protokol kriptosistem apapun.

Kata Kunci: Stream Cipher, Key Stream, Rivest Cipher, RC4, CipherSaber, CS2, IV, KSA, PRGA, XOR.

1. PENDAHULUAN

Sejak zaman Mesir kuno, Yunani Kuno, dan Romawi Kuno, keamanan informasi sudah menjadi suatu hal yang sangat penting. Terutama pada masa peperangan keamanan informasi sering kali menjadi masalah yang sangat krusial, bahkan sebagian besar peperangan berhasil dimenangkan olah salah satu pihak karena pihak yang lain tidak dapat menyimpan informasi rahasianya dengan baik. Dari sanalah muncul ilmu dan seni untuk menjaga keamanan informasi yang disebut sebagai Kriptografi. Sampai saat ini, keamanan informasi masih banyak digunakan dalam menjaga keamanan nasional suatu negara, namun demikian, dewasa ini kriptografi banyak digunakan juga untuk keamanan-keamanan informasi lainnya. Misalnya informasi perusahaan, keuangan, akademis dan informasi-informasi yang sifatnya pribadi.

Pada zaman digital ini, semua data disimpan dan dikirimkan / ditransmisikan dalam bentuk rangkaian (*stream*) *bit* 0 dan 1. Yang membedakan antara suatu data tertentu dengan data yang lain adalah ukuran dari stream bit dan bagaimana 0 dan 1 itu ditempatkan dalam stream bit tersebut. Misalnya data berupa teks dan data yang berupa gambar, dalam data teks suatu rangkaian bit tertentu mewakili satu karakter, sedangkan dalam data gambar suatu rangkaian bit mewakili suatu warna dalam satu *pixel*.

Dalam penyimpanan dan pengiriman data komputer, isi dari data atau informasi tersebut harus dijaga keamanannya. Kerap kali data yang disimpan dalam suatu media penyimpanan bersifat rahasia dan tidak boleh diketahui pihak lain. Apalagi bila data tersebut akan dikirim, informasi di dalam data tersebut harus benar-benar dijaga dan dirahasiakan. Untuk itu, diperlukan proses enkripsi (*encryption*) untuk merahasiakan isi suatu data dan melindungi informasi yang terkandung di dalamnya dari pihak lain dengan merubah isi kode biner yang terkandung dalam data tersebut. Dan juga proses dekripsi (*decryption*) untuk mengembalikan data yang terenkripsi tersebut (cipherteks) menjadi data yang sebenarnya (plainteks).

Ada banyak sekali teori dan metode untuk enkripsi (*cipher*). Salah satu teori yang cukup sederhana tetapi aman adalah RC4 dan CS2. Dalam kedua algoritma enkripsi tersebut, digunakan teori bilangan aritmetika modulo dan konsep logika XOR.

2. SEJARAH RC4

RC4 dikemukakan oleh Ronald Linn Rivest dari Laboratorium RSA (Rivest – Shamir – Adleman) Data Security, Inc. pada tahun 1987. Dari sanalah namanya berasal, RC merupakan singkatan dari *Rivest Cipher* # 4, ada pula beberapa sumber yang mengatakan kepanjangan dari RC adalah *Ron's Code # 4*. Terlepas dari mana yang benar, RC4 merupakan

metode enkripsi stream cipher yang sangat banyak digunakan untuk enkripsi. RC4 diimplementasikan tidak hanya dengan perangkat lunak, tetapi juga dengan perangkat keras.

RC4 ini tidak pernah dipatenkan, dan pada awalnya tidak dipublikasikan dan bersifat rahasia, namun pada tahun 1994 terjadi kebocoran sehingga RC4 tidak lagi menjadi rahasia perusahaan tertentu. Saat ini, RC4 boleh digunakan secara non komersial, namun untuk menghindari masalah *trademark*, nama RC4 diganti dengan ARC4 (*Alleged* RC4) atau ARCFOUR.

3. SEJARAH CS2

Algoritma yang dikemukakan oleh Arnold Reinhold ini dirancang dengan tujuan teknis dan politis. Secara teknis CipherSaber cukup kuat walaupun rancangannya sangat sederhana. Dari segi politis, CipherSaber dibuat karena banyak pemerintahan yang menerapkan larangan penggunaan kriptografi. Dengan rancangan algoritma yang aman namun sangat sederhana ini, Reinhold berharap teknologi enkripsi dapat diimplementasikan dan digunakan dimanapun dan kapanpun oleh semua orang.

Nama CipherSaber berasal dari senjata *light saber* yang digunakan oleh para *Jedi Knight* dalam film *Star Wars*. Menurut Reinhold, nama ini sesuai dengan filosofi alasan perancangan CipherSaber. Berikut adalah kutipan dari *Home Page CipherSaber*:

> *"In George Lucas' Star Wars trilogy, Jedi Knights were expected to make their own light sabers. The message was clear: a warrior confronted by a powerful empire bent on totalitarian control must be self-reliant. As we face a real threat of a ban on the distribution of strong cryptography, in the United States and possibly world-wide, we should emulate the Jedi masters by learning how to build strong cryptography programs all by ourselves. If this can be done, strong cryptography will become impossible to suppress."*

Algoritma Cipher Saber dirancang berdasarkan pada RC4. Keduanya menggunakan larik *State* 256 byte untuk membangkitkan *key stream*. CipherSaber-2 (CS2) dibuat sebagai perbaikan untuk CipherSaber yang pertama ketika Scott Fluhrer, Itsik Mantin, dan Adi Shamir mempublikasikan serangan berbahaya yang dapat dilakukan pada RC4.

4. ALGORITMA RC4

4.1 Teori dan Konsep RC4

RC4 termasuk ke dalam *cipher* aliran (*stream cipher*) dengan kunci privat / kunci simetri (kunci yang sama digunakan untuk proses enkripsi dan dekripsi). Seperti pada *Vernam Cipher*, inti dari enkripsi RC4 adalah pembangkitan kunci aliran (*keystream*) yang bersifat acak semu (*pseudo-random*).

Sebenarnya enkripsi bisa lebih baik bila bilangan yang dihasilkan benar-benar acak (*random*). Namun karena tidak ada komputasi yang benar-benar mengahasilkan deret bilangan acak secara sempurna, bilangan acak yang dihasilkan oleh komputasi adalah bilangan acak semu (*pseudo-random*). Kelemahan bilangan acak semu adalah, pembangkitan bilangannya suatu saat dapat terulang kembali sehingga kemunculannya dapat diprediksi.

Keystream yang dihasilkan nantinya akan dioperasikan dengan operasi logika *exclusive or* $\oplus$ (XOR) dengan plainteks yang akan dienkripsi secara bit per bit (*bit-wise*). Namun tidak seperti Vernam Cipher yang beroperasi pada bit tunggal, RC4 beroperasi secara *byte-wise* pada 1 byte tunggal (8 bit), sehingga enkripsi dilakukan byte per byte. Dengan kata lain RC4 memproses data dengan 1 byte setiap kali transformasi. Dekripsi RC4 dilakukan dengan cara yang sama seperti enkripsinya, hal ini dapat dilakukan karena operasi logika XOR bersifat simetris.

4.1.1. Operasi Logika *exclusive or* (XOR)

Konsep-konsep yang berkaitan dengan operasi logika *exclusive or* diberikan sebagai berikut:

1. Notasi: $\oplus$
2. Operasi:
   $0 \oplus 0 = 0$ $\quad$ $0 \oplus 1 = 1$
   $1 \oplus 0 = 1$ $\quad$ $1 \oplus 1 = 0$
3. Operasi XOR = penjumlahan modulo 2:
   $0 \oplus 0 = 0 \Leftrightarrow 0 + 0 \pmod 2 = 0$
   $0 \oplus 1 = 1 \Leftrightarrow 0 + 1 \pmod 2 = 1$
   $1 \oplus 0 = 1 \Leftrightarrow 1 + 0 \pmod 2 = 1$
   $1 \oplus 1 = 0 \Leftrightarrow 1 + 1 \pmod 2 = 0$
4. Hukum-hukum yang terkait dengan operator XOR:
   (i) $a \oplus a = 0$
   (ii) $a \oplus b = b \oplus a$
   (iii) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
5. Kesimetrisan operasi XOR:
   $(P \oplus K) \oplus K = P$.
   {P = Plainteks, K = Keystream}.
6. Operasi XOR *bitwise*:
   Jika dua rangkaian dioperasikan dengan *XOR*, maka operasinya dilakukan dengan meng-*XOR*-kan setiap bit yang berkoresponden dari kedua rangkaian bit tersebut.
   Contoh: $10011 \oplus 11001 = 01010$

4.1.2. Teori Bilangan – Aritemetika Modulo

Operasi modulo adalah proses mendapakan sisa dari suatu pembagian dari dua bilangan.

1. Misalkan a dan m bilangan bulat ($m > 0$). Operasi ***a* mod *m*** (dibaca "a modulo m") memberikan sisa jika a dibagi dengan m.
2. Notasi: $a \bmod m = r$ sedemikian sehingga $a = mq + r$, dengan $0 \le r < m$.
3. m disebut **modulus** atau **modulo**, dan hasil aritmetika modulo m terletak di dalam himpunan $\{0, 1, 2, \ldots, m-1\}$.
4. Contoh beberapa hasil operasi dengan operator modulo:

(i) 23 mod 5 = 3	(23 = 5 · 4 + 3)
(ii) 27 mod 3 = 0	(27 = 3 · 9 + 0)
(iii) 6 mod 8 = 6	(6 = 8 · 0 + 6)
(iv) 0 mod 12 = 0	(0 = 12 · 0 + 0)
(v) – 41 mod 9 = 4	(–41 = 9 (–5) + 4)
(vi) – 39 mod 13 = 0	(–39 = 13(–3) + 0)

4.2. Algoritma RC4 Secara Umum

Algoritma RC4 cukup sederhana bila dobandingkan dengan algoritma-algoritma enkripsi lain. Implementasinya juga cukup mudah dan hanya terdiri dari beberapa operasi, sehingga RC4 sangat cepat. Secara umum tahap-tahap enkripsi RC4 adalah sebagai berikut:

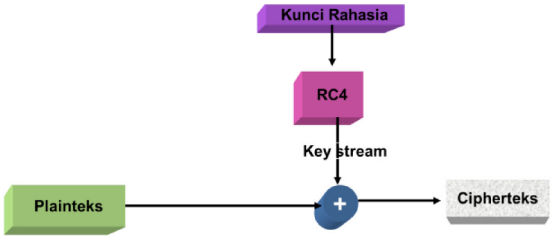
1. Inisialisasi larik *State* sepanjang 256 byte;
2. Memilih Kunci Rahasia yang ingin digunakan (panjang kunci bebas dalam jangkauan antara 1 byte sampai 256 byte, semakin panjang semakin baik);
   Untuk keamanan, lebih baik jangan menggunakan kunci yang pernah digunakan untuk mengenkripsi data yang sama;
   Kunci rahasia ini harus disimpan dengan baik untuk diberikan kepada penerima data untuk keperluan dekripsi;
3. Menjalankan algoritma KSA (*Key-Scheduling Algorithm*) dan PRGA (*Pseudo-Random Generation Algorithm*) untuk membangkitkan bilangan acak semu 8 bit yang menyusun Aliran Kunci (*Key stream*);
4. Key stream yang dihasilkan dioperasikan dengan operasi logika XOR dengan plainteks;
5. Terbentuk cipherteks.

Sedangkan tahap-tahap dekripsinya tidak jauh berbeda dengan tahap-tahap enkripsi:

1. Inisialisasi larik *State* sepanjang 256 byte;
2. Gunakan Kunci Rahasia yang sama seperti yang digunakan dalam proses enkripsi;
3. Bangkitkan Key stream dengan algoritma KSA dan PRGA;
4. Key stream yang dihasilkan dioperasikan dengan operasi logika XOR dengan cipherteks;
5. Plainteks didapatkan kembali.

Untuk membangkitkan aliran kunci, cipher menggunakan status internal yang terdiri dari dua bagian, yaitu:

1. Permutasi angka 0 sampai 255 di dalam larik S_0, S_1, …, S_{255}. Permutasi merupakan fungsi dari kunci U dengan panjang variabel.
2. Dua buah pencacah indeks (iterator), i dan j.



Gambar 1 : Diagram Blok Algoritma RC4 secara umum

4.3. Kode Algoritma RC4 dalam Bahasa Pemrograman

Sebagian besar stream cipher membangkitkan key stream menggunakan LFSR (*Linear Feedback Shift Registers*), yang mangkus jika diimplementasikan sebagai perangkat keras, tapi lambat jika dalam bentuk peragkat lunak. Algoritma RC4 tidak menggunakan LFSR, sehingga sangat ideal untuk implementasi dengan perangkat lunak.

Pertama-tama dilakukan inisialisasi larik *State* (*S*) dengan permutasi identitas. Berikut adalah algoritma yang digunakan untuk menginisialisasi larik *S* dalam bahasa C#:

```
//Inisialisasi Larik State (S):
S = new byte[256];
for(byte i = 0; i < 256; i++)
{
        S[i] = i;
}
```

Setelah diinisialisasi isi larik akan menjadi seperti ini:
S = [0,1,2,3,4,5,6,7,8,9,…,254,255]

Setelah larik *S* diinisialisasi, ditentukan sebuah kunci awal yang sebenarnya panjangnya bebas, namun bila kunci yang dipilih panjangnya kurang dari 256 byte, maka kunci ini akan diduplikasi dan dikonkatenasi terus sampai panjangnya 256 byte (*Key padding*). Algoritma yang dapat digunakan untuk melakukan ini diberikan dalam bahasa C# sebagai berikut:

```
//Masukkan Kunci:
Key = readKey(); //meminta input
kunci awal
//masukkan kunci ke dalam array
U = new byte[256];
for(int u = 0; u < Key.Length; u++)
{
      U[u] = Key[u];
}

//Bila Kunci lebih pendek dari 256
byte, duplikasikan sampai 256 byte:
int iK = Key.Length;
while(iK < 256)
{
      int jK = 0;
      while(jK < Key.Length && iK <
256)
      {
            U[iK] = key[jK];
            iK++;
            jK++;
      }
}
```

Misalnya kunci rahasia yang dipilih adalah:
U = "cipher" (6 byte).
Padding : *U* = "ciphercipher…" sampai panjang *U* mencapai 256 byte.

Setelah itu dilakukan permutasi dari larik *S* yang telah diinisialisasi dengan kunci yang telah ditentukan sebelumnya (kunci ini digunakan untuk mengacak larik), menggunakan KSA (*Key-Scheduling Algorithm*):

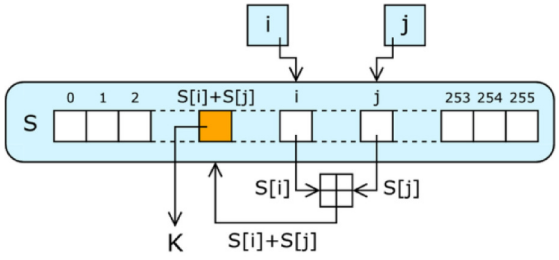
```
//Permutasi:
int jP = 0;
for(byte iP = 0; iP < 256; iP++)
{
      jP = (jP + S[iP] + U[iP]) %
256;
      tukarIsi(S[iP],S[jP]);
}
```

Dengan kunci cipher, setelah KSA, isi larik menjadi:
S = [99,205,63,170,19,138,…,133,57]
Sekarang isi larik *S* sudah teracak berdasarkan kunci cipher.

Setelah permutasi ini selesai dilakukan, hasil permutasi akan digunakan untuk membangkitkan key stream dengan menggunakan algoritma PRGA (*Pseudo-Random Generation Algorithm*). PRGA akan mengganti status dan menghasilkan satu byte key stream setiap kali iterasi. Setiap iterasi, PRGA menginkremen *i*, menambahkan nilai *S* ke-*i* ke *j*, menukar nilai *S*[*i*] dan *S*[*j*], lalu mengeluarkan nilai *S* pada *S*[*i*] + *S*[*j*] (mod 256). Setiap nilai *S* ditukar setidak-tidaknya satu kali setiap 256 kali iterasi. Algoritma PRGA ini diberikan sebagai berikut, juga dalam bahasa C#:

```
int i = 0;
int j = 0;
int idx = 0;
int max = fileIn.BaseStream.Length;
while(index < max) //membaca berkas
masukkan sampai habis
{
      i = (i + 1) % 256;
      j = (j + S[i]) % 256;
      tukarIsi(S[i],S[j]);
      byte t = (S[i] + S[j]) % 256;
      byte K = S[t];
      byte c = K ^
fileIn.ReadByte(); //kunci yang
dibangkitkan di-XOR dengan plainteks
      fileOut.Write(c); //tulis
hasil enkripsi pada berkas keluaran
      idx++;
}
```

Dengan kunci cipher, setelah PRGA, dihasilkan keystream:
77-97-45-32-215-117-61-139-…-16-54
Sekarang keystream telah dibangkitkan berdasarkan kunci cipher.



Gambar 2 : Proses Pembangkitan *Key stream*
(*gambar dari Bahan Kuliah Kriptografi, oleh Ir. Rinaldi Munir, 2006*)

Cipherteks yang dihasilkan dari enkripsi menggunakan RC4 ini, akan berukuran sama dengan plainteksnya.

5. ALGORITMA CS2

5.1 Teori dan Konsep CS2

Seperti halnya RC4, CipherSaber termasuk ke dalam *cipher* aliran (*stream cipher*) dengan kunci privat / kunci simetri (kunci yang sama digunakan untuk proses enkripsi dan dekripsi). Inti dari enkripsi CipherSaber juga berada pada pembangkitan kunci aliran (*keystream*).

Keystream yang dihasilkan nantinya akan dioperasikan dengan operasi logika *exclusive or* ⊕ (XOR) dengan plainteks yang akan dienkripsi secara

bit per bit (*bit-wise*). CipherSaber beroperasi secara *byte-wise* pada 1 byte tunggal (8 bit), sehingga enkripsi dilakukan byte per byte. Seperti RC4, CipherSaber juga memproses data dengan 1 byte setiap kali transformasi. Dekripsi CipherSaber juga dilakukan dengan cara yang sama seperti enkripsinya, hal ini dapat dilakukan karena operasi logika XOR bersifat simetris.

Detail dari operasi logika *exclusive or* (XOR) dan teori bilangan aritmetika modulo telah dipaparkan pada bab sebelumnya.

5.2. Algoritma CS2 Secara Umum

Kesederhanaan algoritma RC4 dimanfaatkan kembali dalam CipherSaber. Implementasinya yang mudah dan jumlah operasinya yang sedikit menjadi alasan penggunaan RC4 sebagai basis CipherSaber. Secara umum tahap-tahap enkripsi CipherSaber adalah sebagai berikut:

1. Inisialisasi larik *State* sepanjang 256 byte (sama dengan RC4);
2. Memilih Kunci Rahasia yang ingin digunakan (panjang kunci bebas dalam jangkauan antara 1 byte sampai 54 byte, semakin panjang semakin baik);
   Tidak seperti RC4, pada CipherSaber, terutama CipherSaber-2, penggunaan kunci yang pernah digunakan untuk mengenkripsi data yang sama, tidak mengurangi keamanannya.
   Hal ini berlaku karena adanya pencampuran kunci dengan IV (*initialization vector*) yang dibangkitkan secara acak (*random*) setiap kali enkripsi. IV ini nantinya disimpan sebagai *header* pada cipherteks;
3. Menjalankan algoritma KSA (*Key-Scheduling Algorithm*) dan PRGA (*Pseudo-Random Generation Algorithm*) untuk membangkitkan bilangan acak semu 8 bit yang menyusun Aliran Kunci (*Key stream*); (sama dengan RC4)
   Pada CipherSaber-2, KSA diulang sebanyak N kali (dengan $N \geq 20$).
4. Key stream yang dihasilkan dioperasikan dengan operasi logika XOR dengan plainteks (sama dengan RC4);
5. Terbentuk cipherteks.

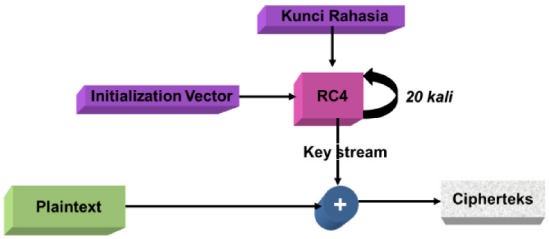
Sedangkan tahap-tahap dekripsinya tidak jauh berbeda dengan tahap-tahap enkripsi:

1. Inisialisasi larik *State* sepanjang 256 byte (sama dengan RC4);
2. Gunakan Kunci Rahasia yang sama seperti yang digunakan dalam proses enkripsi (sama dengan RC4);
3. Membaca IV dari *header* cipherteks untuk dicampurkan kembali dengan kunci;
4. Bangkitkan Key stream dengan algoritma KSA dan PRGA (sama dengan RC4);
5. Key stream yang dihasilkan dioperasikan dengan operasi logika XOR dengan cipherteks (sama dengan RC4);
6. Plainteks didapatkan kembali.

Untuk membangkitkan aliran kunci, cipher menggunakan status internal yang terdiri dari dua bagian, yaitu:

1. Permutasi angka 0 sampai 255 di dalam larik S_0, S_1, …, S_{255}. Permutasi merupakan fungsi dari kunci U dengan panjang variabel.
2. Dua buah pencacah indeks (iterator), i dan j.

Diagram blok algoritma CipherSaber secara umum mirip dengan diagram blok algoritma RC4:



Gambar 3 : Diagram Blok Algoritma CipherSaber-2 secara umum

5.3. Kode Algoritma CS2 dalam Bahasa Pemrograman

Sama seperti algoritma RC4, algoritma CipherSaber juga tidak menggunakan LFSR (*Linear Feedback Shift Register*), layaknya cipher aliran pada umumnya, sehingga sangat ideal untuk implementasi dengan perangkat lunak.

Pertama-tama dilakukan inisialisasi larik *State* (*S*) dengan permutasi identitas. Berikut adalah algoritma yang digunakan untuk menginisialisasi larik *S* dalam bahasa C#:

```
//Inisialisasi Larik State (S):
S = new byte[256];
for(byte i = 0; i < 256; i++)
{
        S[i] = i;
}
```

Langkah ini sama seperti langkah awal pada algoritma RC4. Setelah diinisialisasi isi larik akan menjadi seperti ini:

S = [0,1,2,3,4,5,6,7,8,9,…,254,255]

Setelah larik *S* diinisialisasi, ditentukan sebuah kunci awal yang sebenarnya panjangnya bebas, dengan jangkauan antara 1 byte sampai 54 byte. Panjang kunci maksimal 54 byte agar kunci tersebut

dapat tercampur dengan baik dengan IV (*initialization vector*) yang dibangkitkan secara random pada setiap kali enkripsi. Selanjutnya gabungan dari kunci dan IV ini akan diduplikasi dan dikonkatenasi terus sampai panjangnya 256 byte (*Key padding*). Algoritma yang dapat digunakan untuk melakukan ini diberikan dalam bahasa C# sebagai berikut:

```
//Masukkan Kunci:
Key = readKey(); //meminta input
kunci awal
//masukkan kunci ke dalam array
U = new byte[256];
for(int u = 0; u < Key.Length; u++)
{
      U[u] = Key[u];
}
//masukkan IV ke dalam array
int iv = 0;
for(int i = Key.Length; i <
Key.Length+10; i++)
{
      //bila enkripsi, IV
dibangkitkan secara random:
      if (encrypt) IV[iv] =
random.Next(0,256);
      //bila dekripsi, IV dibaca
dari header cipherteks:
      else IV[iv] =
fileIn.ReadByte();
      U[i] = IV[iv];
      iv++;
}

//Duplikasikan sampai 256 byte:
int iK = Key.Length+10;
while(iK < 256)
{
      int jK = 0;
      while(jK < Key.Length && iK <
256)
      {
            U[iK] = key[jK];
            iK++;
            jK++;
      }
}
```

Pada tahap inilah perbedaan antara RC4 dengan CipherSaber jelas terlihat. CipherSaber menggunakan IV (*initialization vector*) yang dicampurkan dengan kunci awal. Pencampuran ini dilakukan untuk menyembunyikan pola statistik yang dapat terjadi bila kunci yang sama digunakan berulang-ulang.

Sama seperti pada RC4, setelah itu dilakukan permutasi dari larik *S* yang telah diinisialisasi dengan kunci yang telah ditentukan sebelumnya (kunci ini digunakan untuk mengacak larik), menggunakan KSA (*Key-Scheduling Algorithm*) / *mixing operation*:

```
//Permutasi dan Pencampuran Kunci:
int jP = 0;
for(byte iP = 0; iP < 256; iP++)
{
      jP = (jP + S[iP] + U[iP]) %
256;
      tukarIsi(S[iP],S[jP]);
}
```

Pada operasi *mixing* tersebut, larik *State* diacak dengan 256 kali operasi *mixing*. Tahap-tahap operasi ini adalah sebagai berikut:

1. Pada variabel *jP* ditambahkan elemen kunci yang ke-iP dan elemen *State* yang ke-*iP*.
2. Tukar nilai *State* yang ke-*iP* dan yang ke-*jP*.

Seperti yang sudah dijelaskan sebelumnya, pada CipherSaber-2 operasi *mixing* ini dilakukan sebanyak *N*-kali (dengan $N \geq 20$).

Setelah permutasi ini selesai dilakukan, hasil permutasi akan digunakan untuk membangkitkan key stream dengan menggunakan PRGA. Generator kunci akan mengganti status dan menghasilkan satu byte key stream pada setiap iterasi. Untuk setiap byte data (iterasi) lakukan langkah-langkah sebagai berikut:

1. Inkremen *i*.
2. *j* ditambahkan dengan elemen *State* yang ke-*i*
3. Tukar nilai elemen ke-*i* dengan elemen ke-*j* dari *State*.
4. Jumlahkan nilai elemen ke-*i* dan ke-*j* dari *State*.
5. Bangkitkan sebuah byte key stream dari elemen ke-*t*.
6. Byte key stream yang dihasilkan di-XOR-kan dengan byte masukkan, dan hasilnya dituliskan ke keluaran.

PRGA diberikan sebagai berikut, juga dalam bahasa C#:

```
int i = 0;
int j = 0;
int idx = 0;
int max = fileIn.BaseStream.Length;
while(index < max) //membaca berkas
masukkan sampai habis
{
      i = (i + 1) % 256;
      j = (j + S[i]) % 256;
      tukarIsi(S[i],S[j]);
      byte t = (S[i] + S[j]) % 256;
      byte K = S[t];
      byte c = K ^
fileIn.ReadByte(); //kunci yang
dibangkitkan di-XOR dengan plainteks
      fileOut.Write(c); //tulis
hasil enkripsi pada berkas keluaran
      idx++;
}
```

Cipherteks yang dihasilkan dari enkripsi menggunakan CipherSaber ini, akan berukuran sama

dengan plainteksnya, ditambah 10 byte *header* yang berisi IV (*initialization vector*).

6. PERBANDINGAN RC4 DAN CS2

CipherSaber dirancang berdasarkan pada RC4 sehingga keduanya tidak banyak berbeda. Perbandingan algoritma RC4 dan CipherSaber-2 diberikan pada tabel berikut ini:

Tabel 1 Perbandingan Algoritma RC4 dan CipherSaber-2

	RC4	CipherSaber-2
1	State Initialization	
2	Key Padding	Sebelum di-padding, kunci ditambahkan dengan random Initialization Vector
3	Key Scheduling Algorithm	Key Scheduling Algorithm diulang sebanyak N kali dengan $N \geq 20$
4	Pseudo-Random Generation Algorithm	
5	Input $\oplus$ Key Stream	

Detail dari tiap langkah pada tabel di atas telah diberikan pada bab sebelumnya.

Pencampuran random *initialization vector* pada key stream bermanfaat bila kunci yang digunakan sama untuk beberapa kali enkripsi. Sedangkan pengulangan *Key Scheduling Algorithm* dilkukan untuk meningkatkan keamanannya. Dengan melakukan pencampuran dan pengulangan ini, input dari PRGA RC4 seakan-akan telah melalui *preprocessing* oleh algoritma yang kompleks.

Waktu yang diperlukan untuk enkripsi arsip yang sama dengan menggunakan CipherSaber-2 secara umum sama dengan RC4. Berikut adalah rincian pengujian waktu yang diperlukan untuk oleh kedua algoritma.

Tabel 2 Kecepatan enkripsi RC4

	Ukuran Arsip (byte)	Waktu yang Diperlukan (detik)	Kecepatan Enkripsi (byte/detik)
1	162	0,003	54 000
2	1 785	0,014	127 500
3	24 245	0,153	158 464
4	437 992	2,591	169 043
5	4 497 171	26,263	171 235

Tabel 3 Kecepatan enkripsi CipherSaber-2

	Ukuran Arsip (byte)	Waktu yang Diperlukan (detik)	Kecepatan Enkripsi (byte/detik)
1	162	0,003	54 000
2	1 785	0,014	127 500
3	24 245	0,152	159 506
4	437 992	2,577	169 961
5	4 497 171	26,311	170 923

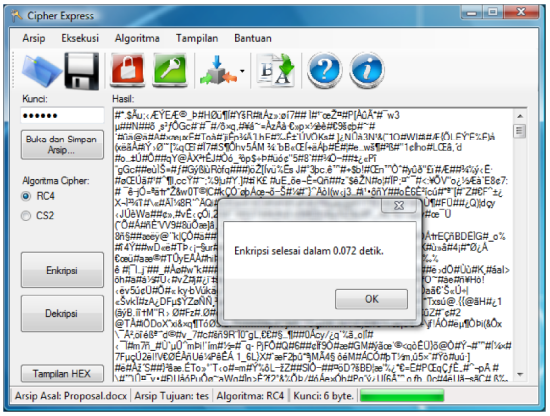
Pengukuran kecepatan dilakukan dengan menggunakan prosesor Intel Core Duo T2500 @ 2.0 GHz x 2, 533.5 MHz Rated FSB, 2 MB L2 Cache, dengan 2560 MB Memory.

Dilihat dari proses komputasi yang dilakukan oleh kedua algoritma, CipherSaber-2 seharusnya memiliki waktu komputasi yang sedikit lebih lama dibanding RC4. Hal ini disebabkan oleh adanya pencampuran antara kunci dan IV (*initialization vector*) dan terutama pengulangan KSA sebanyak N kali yang dilakukan pada CipherSaber-2. Namun karena pengukuran hanya dilakukan pada arsip-arsip berukuran kecil dan ketelitian pengukuran hanya sampai tiga angka di belakang koma, perbedaan tersebut kurang terlihat pada hasil pengujian kecepatan.

Pengubahan 1 bit pada cipherteks kedua algoritma (baik RC4 maupun CipherSaber-2) hanya mempengaruhi 1 byte pemilik bit yang bersangkutan pada plainteks hasil dekripsi. Sedangkan penambahan / penghilangan satu byte atau lebih pada cipherteks kedua algoritma (baik RC4 maupun CipherSaber-2) akan merusak seluruh stream setelah posisi penambahan / pengurangan tersebut pada plainteks hasil dekripsi.

7. IMPLEMENTASI RC4 DAN CS2

Bersamaan dengan penulisan makalah ini, telah dibuat sebuah program aplikasi sederhana yang dapat mengenkripsi arsip apapun dengan menggunakan algoritma RC4 standar atau CS2. Program aplikasi tersebut dibuat dengan bahasa pemrograman C# di atas platform Microsoft .NET Framework 3.5 Service Pack 1. Program ini digunakan untuk melakukan beberapa pengujian dengan parameter uji sederhana (misalnya *random level* (seacak apa cipherteks hasil dekripsi) *speed*, *bit flip*, *pseudo-byte addition*, dan *byte removal*).



Gambar 4 : Program aplikasi pengujian

(Selain program aplikasi biasa, pada awalnya akan dibuat juga sebuah Windows Vista Sidebar Gadget. Dengan Gadget ini pengguna dapat dengan lebih mudah mengenkripsi atau mendekripsi arsip apapun (enkriptor dapat diakses langsung di desktop tanpa harus membuka aplikasi tertentu). Sidebar Gadget tersebut rencananya akan dibuat dengan kakas Script# (kakas pada Microsoft Visual Studio untuk mengkonversi C# menjadi JavaScript). Rencana pambuatan Sidebar Gadget hanya sebagai *supplementary* makalah ini dan tidak digunakan untuk melakukan pengujian. Namun demikian, karena keterbatasan waktu, Windows Vista Sidebar Gadget tersebut belum sempat diselesaikan bersamaan dengan selesainya penulisan makalah ini, sebagaimana yang dituliskan dalam proposal penulisan makalah).

Seperti yang telah dijelaskan pada bab-bab yang lalu, implementasi kedua alogritma stream cipher ini cukup sederhana, terutama implementasi algoritma CipherSaber yang merupakan modifikasi dari RC4. Tetapi walaupun implementasinya sederhana, kedua algoritma ini masih diakui kekuatannya, bahkan salah satunya masih digunakan secara luas dalam banyak bidang keamanan informasi.

Program aplikasi desktop yang dibuat bisa didapatkan dengan mengunduh pada pranala berikut ini:

http://students.if.itb.ac.id/~if17034/crypto/

atau melalui e-mail penulis.

8. KEKUATAN DAN KELEMAHAN RC4

8.1. Kekuatan RC4

Kehebatan RC4 yang sudah pasti, adalah kekuatannya, padahal metode ini juga terkenal karena kesederhanaan dan kecepatannya (enkripsi dengan RC4 kira-kira 5 kali lebih cepat daripada enkripsi dengan DES (*Data Encryption Standard*) dan 15 kali lebih cepat daripada enkripsi dengan Triple DES).

Pembangkitan key stream pseudo-random-nya pun hampir mendekati One-Time-Pad yang konon *unbreakable*. RSA bahkan mengklaim bahwa RC4 ini kebal terhadap kriptanalisis diferensial dan linear. Dengan demikian RC4 ini boleh dikatakan cukup kuat.

8.2. Kelemahan RC4

Walaupun kuat, sederhana, dan cepat, RC4 memiliki kelemahan bila kunci yang digunakan pendek. Dan satu setiap 256 kunci dapat menjadi kunci yang lemah. Kunci-kunci lemah ini dapat diidentifikasi oleh para kriptanalis. Menurut beberapa peneliti, byte-byte pertama dalam key stream tidak benar-benar acak, dan dapat digunakan untuk melacak kunci. Tetapi, untuk mengatasi masalah ini, bagian depan key stream yang dihasilkan untuk enkripsi / dekripsi dapat dibuang untuk menghindari serangan.

Saat kunci didupilkasi sampai 256 byte (*padding*), dapat menyebabkan adanya nilai-nilai yang sama di dalam larik S. Dan seperti stream cipher yang lain, RC4 mudah diserang dengan cara mengoperasikan dengan operasi logika *exclusive or* ⊕ (XOR) dua set byte dari cipherteks. Metode ini dikenal dengan nama *known-plaintext attack*. Selain itu juga ada beberapa serangan lainnya terhadap RC4 yaitu, Fluhrer, Mantin, and Shamir Attack (2001) yang membobol enkripsi RC4 pada WEP (*Wired Equivalent Privacy*) serta Andreas Klein's Attack (2005) yang berhasil mendapatkan korelasi antara key stream dengan kunci awal dan berhasil membobol RC4 pada WEP dalam waktu kurang dari 1 menit. Karena banyaknya celah keamanan RC4 pada WEP, sekarang banyak yang beralih ke WPA (*Wi-Fi Protect Access*) yang juga menggunakan RC4. Belakangan ini juga ditemukan kelmahan-kelemahan lain pada Keystream Generator RC4.

9. KEKUATAN DAN KELEMAHAN CS2

9.1. Kekuatan CS2

Karena dirancang dengan basis RC4, CipherSaber mewarisi kekuatan-kekuatan dan kelemahan-kelamahan yang dimiliki RC4. CipherSaber cukup kuat, padahal implementasi metode ini sangat sederhana. Selain itu, algoritma ini dirancang supaya cipherteks yang dihasilkan tidak dapat dibedakan dengan *random noise*.

Untuk mengatasi kelemahan RC4 dimana bila pesan dalam jumlah besar dikirim dengan kunci yang sama, dibuat perbaikan dari CipherSaber yang disebut CipherSaber-2.

9.2. Kelemahan CS2

CipherSaber cukup kuat dan berguna untuk membuat tujuan politisnya efektif. Namun demikian,

keamanan dan kenyamanan penggunaannya tidak sepopuler kriptosistem yang lain. Kelemahan-kelamahan CipherSaber antara lain:

- CipherSaber tidak menyediakan otentikasi pesan (Penyerang dapat mengubah isi cipherteks asli dengan pesan lain yang panjangnya sama, sehingga maknanya berubah).
- CipherSaber tidak memiliki fitur manajemen kunci (Bagaimana memberikan kunci dekripsi kepada penerima pesan dengan aman?).
- Perbaikan pada CipherSaber-2 belum terbukti kefektifannya.
- CipherSaber lemah terhadap *dictionary attack* bila kunci yang digunakan lemah.
- CipherSaber tidak menangani serangan terhadap deteksi cipherteks (Tidak menangani steganografi).

10. PENERAPAN RC4

Enkripsi RC4 digunakan pada banyak sistem keamanan seperti SSL (*Secure Socket Layer*) untuk menjaga keamanan lalu lintas data di internet, TLS (*Transport Layer Security*), TKIP (*Temporal Key Integerity Protocol*) dan sistem keamanan yang sudah kita kenal baik pada koneksi nirkabel (*wireless*) IEEE 802.11, yaitu WEP (*Wired Equivalent Privacy*) dan WPA (*Wi-Fi Protect Access*). Karena kekuatan dan kesederhanaanya, 99,03 % server di dunia sudah mendukung RC4.

WEP menggunakan RC4 pada modus sinkron untuk mengekripsi paket-paket data secara *byte-wise*. Pada enkripsi stream cipher seperti RC4, bila ada 1 bit saja yang hilang ketika dienkripsi akan mengakibatkan hilangnya seluruh data setelah 1 bit data yang hilang tersebut (termasuk data pada paket berikutnya). Ini sebenarnya merupakan masalah pada WEP, karena WEP menggunakan enkripsi yang tidak cocok. Perlu ditekankan di sini bahwa masalahnya tidak terletak pada algoritma RC4. Masalahnya adalah stream cipher tidak cocok untuk medium nirkabel. Ini berkebalikan dengan SSL.

Walaupun demikian IEEE (*Institute of Electrical and Electronics Engineers*) tetap memilih untuk menggunakan RC4 yang kurang cocok sebagai enkripsi nirkabel dibandingkan dengan AES (*Advanced Encryption Standard*) yang lebih cocok karena AES membutuhkan lebih banyak kinerja dibandingkan dengan yang didukung oleh kartu *wireless adapter* 802.11 yang ada saat ini.

Selain pada keamanan lalu lintas data nirkabel, RC4 juga digunakan pada BitTorrent *protocol encryption*, MPPE (Microsoft *Point to Point Encryption*), Secure Shell, *Remote Desktop Protocol*, *Kerberos Protocol*, SASL (*Simple Authentication and Security Layer*), Gpcode.AK (virus computer yang menyerang Microsoft Windows, yang mengambil dokumen dengan mengenkripsinya menggunakan RC4 dan RSA-1024. Virus ini kemudian meminta tebusan untuk dokumen yang diambilnya), Microsoft XBOX, Oracle Secure SQL, Microsoft PPTP, Microsoft Office, Adobe Acrobat PDF (*Portable Document Format*), dan Lotus Notes.

11. PENERAPAN CS2

Walaupun cukup kuat dan implementasinya sangat sederhana, karena kelemahan-kelemahannya, CipherSaber dianggap tidak cocok untuk penggunaan luas dan tidak banyak digunakan seperti RC4. Biasanya CipherSaber banyak digunakan oleh pengguna-pengguna awam untuk bertukar pesan pribadi.

12. KESIMPULAN

Keamanan informasi sangat diperlukan pada dunia komputasi, terutama dewasa ini di mana informasi sangat berpengaruh bagi keamanan nasioal, keuangan negara, keuangan perusahaan, data akademis, data-data pribadi, dan sebagainya. Dengan demikian diperlukan suatu metode pengamanan yang efektif.

Salah satu metode enkripsi yang kuat namun sederhana adalah enkripsi dengan menggunakan algoritma Rivest Cipher 4 (RC4). Algoritma RC4 meliputi inisialisasi larik, *key padding*, KSA (*Key-Scheduling Algorithm*), PRGA (*Pseudo-Random Generation Algorithm*), dan operasi XOR (*exclusive or*). Algoritma RC4 menerima kunci rahasia pilihan pengguna yang panjangnya fleksibel (antara 8 bit sampai 2048 bit), menghasilkan *keystream*, dan meng-XOR-kan keystream tersebut dengan plainteks untuk menghasilkan cipherteks. Cipherteks hasil enkripsi algoritma RC4 dapat didekripsikan dengan cara yang sama dengan enkripsinya.

Metode enkripsi lain yang mirip dengan RC4 adalah CipherSaber. CipherSaber yang dirancang berdasarkan pada RC4 ini, sangat sederhana namun cukup kuat. Algoritma CipherSaber memiliki beberapa langkah yang serupa dengan RC4. Algoritma CipherSaber meliputi inisialisasi larik, *key padding*, KSA (*Key-Scheduling Algorithm*), PRGA (*Pseudo-Random Generation Algorithm*), dan operasi XOR (*exclusive or*). Seperti RC4, algoritma CipherSaber juga menerima kunci rahasia pilihan pengguna yang panjangnya fleksibel (antara 8 bit sampai 432 bit), menghasilkan *keystream*, dan meng-XOR-kan keystream tersebut dengan plainteks untuk menghasilkan cipherteks. Cipherteks hasil enkripsi algoritma CipherSaber juga dapat didekripsikan dengan cara yang sama dengan enkripsinya.

Perbedaan utama dari RC4 standar dan CipherSaber-2 adalah adanya pencampuran kunci dengan IV (*initialization vector*), serta pengulangan

KSA sebanyak N (dengan N ≥ 20) kali pada CipherSaber-2.

Dari segi keamanan cipherteks, kedua algoritma cukup aman dari serangan bit-flip attack karena pengubahan 1 bit pada cipherteks hanya merusak byte pemilik bit tersebut pada plainteks hasil dekripsi. Sedangkan pengurangan atau penambahan byte berakibat cukup fatal pada kedua algoritma. Semua stream plainteks hasil dekripsi pada posisi setelah penambahan / pengurangan semuanya rusak.

Dari segi kekuatan, CipherSaber-2 terlihat lebih aman dibandingkan dengan RC4 standar. Namun demikian, RC4 ternyata lebih formal, dan didukung oleh banyak protokol yang memiliki lebih banyak fitur sehingga lebih populer digunakan dalam berbagai bidang.

Saat ini RC4 banyak digunakan dalam berbagai keamanan untuk berbagai aplikasi dan aliran data. Terutama aliran data nirkabel (*wireless*). Sedangkan CipherSaber tidak digunakan secara formal, penggunaan CipherSaber hanya sebatas pada pertukaran informasi pribadi oleh orang-orang pecinta kriptografi.

REFERENSI

[1] Bahan kuliah Kriptografi oleh Ir. Rinaldi Munir:
Algoritma Kriptografi Modern (Bagian 1), 2005;
Algoritma Kriptografi Modern (Bagian 2), 2005;
Pembangkit Bilangan Acak Semu (Bagian 1), 2006;
Pengatar Kriptografi, 2006;
RC4 dan A5, 2006

[2] Bahan kuliah Struktur Diskrit oleh Ir. Rinaldi Munir:
Teori Bilangan, 2005

[3] Erich Nahum, *Cryptographic Strength of SSL/TLS Servers*, IBM Research, 2002

[4] Eli Biham, Louis Granboulan, dan Phong Nguyễn, *Impossible Fault and Differential Fault Analysis of RC4*, Bruge, 2004

[5] Misbah Rehman, *RC4 Stream Cipher*, California State University, 2005

[6] Vivek Ramachandran, *Encryption/Decryption using RC4*, Airtight, 2007

[7] "802.11 Security", Wireless Developer Network: http://www.wirelessdevnet.com

[8] "CipherSaber", CipherSaber Home Page: http://ciphersaber.gurus.org

[9] "CipherSaber Hints and Tips", Mr. Tines: http://www.ravnaandtines.com

[10] "RC4", The Free Dictionary: http://encyclopedia2.thefreedictionary.com

[11] "RC4 Encryption Algorithm", Vocal: http://www.vocal.com

[12] "What is RC4?", Search Security: http://searchsecurity.techtarget.com

[13] "What is RC4?", Tech FAQ: http://www.tech-faq.com