Analisa Mendalam Pustaka Pembangkit Bilangan Acak Semu Pada Linux dan Windows

Mohamad Octamanullah
NIM 13503119

Abstrak

Hampir semua aspek kriptografi berhubungan dengan bagiamana membangkitkan bilangan acak. Untuk membangkitkan bilangan yang benar-benar acak dibutuhkan perangkat keras yang dapat menangkap sinyal dari indra eksternal untuk menentukan bilangan acak, seperti: memperhatikan radioaktif atom, noise dari suara, gambar dari kamera, dll. Namun perangkat keras tersebut tidak hanya mahal, namun juga kurang efektif untuk digunakan dalam kebutuhan kriptografi secara umum. Untuk itu dibutuhkan suatu algoritma dan perangkat lunak yang dapat membangkitkan bilangan acak, namun pada aplikasinya, bilangan acak yang dibangkitkan baik oleh perangkat lunak merupakan bilangan acak semu. Banyak algoritma yang dikemukakan untuk membangkitkan bilangan acak semu. Pada paper ini akan dibahas pustaka pembangkit bilangan semu pada sistem operasi Linux dan Microsoft Windows. Kedua sistem operasi tersebut merupakan sistem operasi yang memiliki pustaka pembangkit bilangan acak semu yang tersedia secara *native* dan telah *built in* pada kernel.

Di level diatas tersebut terdapat banyak pustaka-pustaka independen yang juga memiliki fungsionalitas membangkitkan bilangan acak. Pada paper ini akan dibahas beberapa pustaka umum yang dapat ditemui tentang perilaku pembangkitan bilangan acak, dan bagaimana kelebihan dan kekuarangannya, serta keterhubungannya dengan API native. Di paper ini akan dibahas pula pembangkit bilangan acak yang bergantung pada CPU, sebuah pernagkat yang selalu ada di setiap komputer. Pada beberapa keluarga CPU, mereka memiliki fungsionalitas pembangkit bilangan acak yang bersumber pada penangkapan suhu prosesor.

Kata Kunci: Bilangan Acak, Bilangan Acak Semu, Sistem Operasi, Pembangkit Bilangan Acak. API pembangkit bilangan acak.

1. Pendahuluan

Pengacakan adalah suatu masalah yang sangat krusial dalam bidang kriptografi dan pada bidang-bidang lain seperti statistik, simulasi komputer, perjudian, dll. Dalam bidang kriptografi, dikarenakan sangat pentingnya bilangan acak, maka bilangan acak menem[ati tempat yang cukup banyak dalam penelitian di bidang tersebut. Dengan menyerang pembangkit bilangan acak maka dengan sendirinya menyerang sistem kriptografi secara umum, sebab dengan mengetahui sekuen angka acak yang akan dihasilkan maka akan dapat dengan mudah mengetahui hasil enkripsi dan sekripsi suatu algoritma kriptografi.

Dalam proses menganalisa sistem keamanan suatu sistem, salah satunya adalah dengan menganalisa pembangkit bilangan acaknya, yaitu sumber masukan pembangkit dan hasil keluaran bilangan acak tersebut. Pembangkit bilangan acak yang lemah ditandai dengan mudahnya diprediksi angka yang akan keluar, yang akan menyebabkan mudahnya menjebol sistem secara keseluruhan. Hal ini dicontohkan pada kasus penjebolan implementasi Secure Socket Layer (SSL) pada aplikasi Netscape [4], sebuah protokol komunikasi yang sangat aman, namun lemah karena pembangkit bilangan acaknya. Atau pada kasus memprediksi Java session-id [5], yang dengan upaya yang tidak susah dapat menerka session-id dari sebuah session aplikasi Java.

Pembangkit bilangan acak sebenarnya sangat mudah dengan cara melihat proses fisik, semisal suara yang ditangkap oleh microfon, gambar yang ditangkap oleh web camera, dll. Namun untuk mengimplementasi hal tersebut terbilang mahal dalam percakapan biaya komputasi. Untuk itu setiap sistem pasti memiliki cara untuk menciptakan bilangan acak semu. Semu sebab bukan benar-benar suatu bilangan acak, namun sebuah bilangan yang

dihasilkan dari hasil komputasi terhadap data-data yang dianggap acak. Data yang dianggap acak tersebut disebut entropi, yang juga dikumpulkan dari perangkat keras yang terhubung ke sistem.

State pertama dari sebuah pembangkit bilangan acak adalah menanam biji (*seed*) bilangan acak. Dengan menanam biji yang sama, maka pohon bilangan acak yang dihasilkan juga akan sama. Hal ini berguna bagi banyak hal di berbagai bidang.

State berikutnya adalah menyiram tanaman tersebut dengan entropi-entropi yang berhasil dikumpulkan. Dengan menyiram tersebut diharapkan timbul bilangan acak baru.

State terakhir adalah memperbarui buah angka bilangan acak, yaitu dengan proses komputasi menggunakan algoritma pembangkit bilangan acak khusus yang akan selalu memperbarui bit-bit pada pembangkit bilangan acak.

Pembangkit bilangan acak juga diharuskan aman dari berbagai macam serangan, baik serangan eksternal maupun serangan internal. Penyerang diasumsikan mengetahui keseluruhan atau setidaknya sebagian kode pembangkit bilangan acak. Penyerang juga diasumsikan mengetahui entropi-entropi yang digunakan sebagai sumber data pembangkit bilangan acak. Dibawah ini adalah list kebutuhan dasar dari keamanan terhadap pembangkit bilangana acak menggunakan terminologi umum [6] (list celah keamanan yang lebih detail dapat dilihat pada [7]):

- Keacakan. Pembangkit bilangan acak menghasilkan kumpulan bilangan yang terlihat benar-benar acak dari luar.
- Keamanan prediksi. Walaupun terdapat pengetahuan state dari pembangkit bilangan acak pada suatu saat, mulai dari entropi hingga keluarannya, namun tidak dapat memprediksi apapun khususnya keluaran selanjutnya dari pembangkit bilangan acak.
- Keamanan masa lalu. Walaupun terdapat pengetahuan state dari pembangkit bilangan acak pada suatu saat khususnya jika diberikan sejumlah entropi yang sama persis tidak akan menyebabkan keluran pembangkit bilangan acak menjadi sama.

Kebutuhan akan sifat keacakan sebenarnya cukup untuk mengatasi skenario serangan umum, sebab pelaku penyerangan hampir tidak mungkin mengetahui status dari pembangkit bilangan acak. Tapi dalam beberapa skenario khusus yang jarang terjadi, pelaku penyerangan dapat mengetahui state dari pembangkit tersebut, seperti contoh: dengan melewati restriksi akses dari sistem operasi dapat mengetahui state dari pembangkit bilangan acak, atau dengan membaca langsung nilai di memori atau di hard disk bila tersimpan di sana.

2. Kebutuhan Pembangkitan Bilangan Acak Semu

Pembangkitan bilangan acak dubutuhkan oleh berbagai hal. Untuk itu diperlukan sebuah alat yang dapat mensuplai bilangan acak. Terdapat banyak perangkat keras yang mampu memberikan bilangan acak berdasarkan sumber eskternal, seperti mendengarkan suara dari mikrofon, memperhatikan gambar dari web cam, dll. Namun peralatan tersebut sangat mahal dalam hal komputasi. Sehingga dibutuhkan suatu kemampuan pembangkitan bilangan dalam bentuk algoritma dan perangkat lunak. Walaupun menurut pendapat John Von Neumann yang berkata “Seseorang yang menggunakan metoda aritmatika dalam memproduksi angka-angka acak adalah suatu perbuatan dosa”.

Pembangkitan bilangan acak sangat dibutuhkan oleh berbagai bidang keilmuan. Namun dua bidang keilmuan yang sangat bergantung pada pembangkitan bilangan acak adalah bidang statistik dan kriptografi.

Pada bidang statistik, bilangan acak dibutuhkan untuk menghasilkan suatu analisa statistik, untuk itu bilangan acak harus benar-benar acak. Bila bilangan acak memiliki sifat-sifat tertentu karena hasil perhitungan aritmatika tertentu, misal: berulang, maka hasil analisas statistiuk yang dihasilkan tidak akurat dan kurang dapat dipercaya.

Pada bidang kriptografi, bilangan acak dibutuhkan untuk menghasilkan berbagai hal, seperti menghasilkan kata kunci yang baik, menghasilkan kunci dan padanannya pada algoritma kriptografi asimetrik/publik, membangkitkan vektor inisialisasi pada hampir seluruh algoritma kriptografi, dll.

Pelaku kejahatan dalam usaha memecahkan algoritma kriptografi tertentu biasanya juga berusaha memecahkan pembangkit bilangan acak. Dengan memecahkan pembangkit bilangan acak, seperti membuat agar pembangkit bilangan acak selalu menghasilkan angka yang sama, maka secara keseluruhan telah berhasil memecahkan keamanan kriptografi.

3. Langkah-langkah Dasar Pembangkitan Bilangan Acak Semu

3.1 Struktur Umum Pembangkit Bilangan Acak Semu

Struktur umum pembangkitan bilangan acak semu terdiri atas tiga bagian utama. Pada tahap pertama, entropi sistem operasi mulai dikumpulkan dari berbagai kejadian (*events*) yang ditangkap oleh kernel.

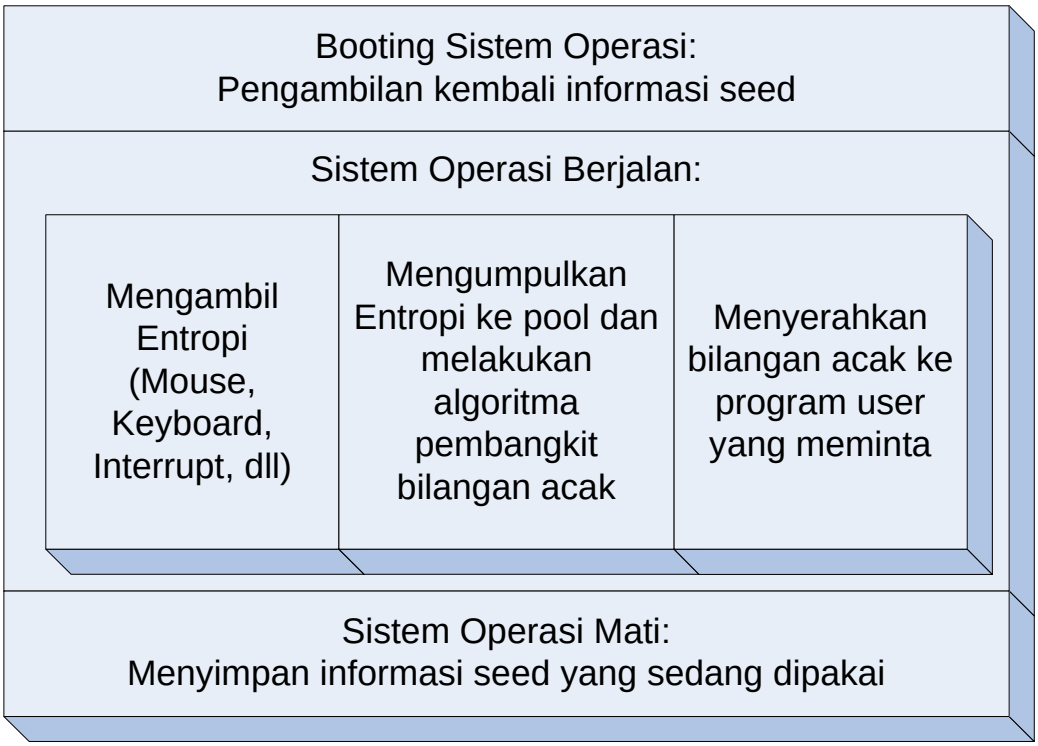
Pada tahap kedua, entropi yang berhasil dikumpulkan diberikan kepada sebuah entitas penampung (*pool*). Entitas ini akan melakukan pencampuran dan penanganan kejadian yang diberikan oleh sistem operasi.

Tahap terakhir terjadi bila bilangan acak tersebut dibutuhkan oleh pengguna. Bilangan acak diberikan kepada pengguna dengan melalui beberapa algoritma pembangkit bilangan. Setelah bilangan acak diberikan, dimulai lagi tahap pertama.

Satu-satunya proses yang menggunakan operasi kriptografi non-linear adalah fungsi pembangkit hash SHA-1. (Walaupun terdapat banyak paper yang membahas ketidakamanan fungsi hash SHA-1, namun paper ini tidak akan membahas hal tersebut).

Pada kedua sistem operasi (Microsoft Windows dan Linux), terdapat dua tipe pembangkit bilangan acak, yaitu: pembangkit bilangan acak yang bersifat *blocking* dan pembangkit bilangan acak yang tidak bersifat *blocking*. Yang dimaksud dengan blocking adalah terjadinya penungguan dalam program user mana kala sedang meminta sebuah bilangan acak. Sebaliknya non-blocking berarti program user tidak perlu menunggu dalam proses meminta bilangan acak.

Proses blocking terjadi dikarenakan sistem operasi membutuhkan waktu dalam memproses entropi-entropi yang dimilikinya dan dalam menentukan angka selanjutnya yang akan diberikan kepada pengguna. Dalam sistem blocking, program user menginginkan angka yang diberikan benar-benar telah melalui proses pembangkitan bilangan acak yang baik.



Sebaliknya pada proses non blocking, sistem operasi akan memberikan kepada user angka yang sedang dimilikinya saat ini, walaupun proses pembangkitan bilangan acak belum selesai. Program user juga seharusnya sadar akan hal ini, sehingga dalam proses penggunaan bilangan tersebut juga tidak boleh pada kasus-kasus yang sarat dengan keamanan.

3.2 Inisialisasi

Pada saat proses pemulaian (*start-up*), setiap sistem operasi melakukan berbagai kegiatan rutin. Kegiatan-kegiatan tersebut salah satunya adalah menginisialisasi pembangkit bilangan acak dengan parameter konstan sistem operasi. Parameter tersebut antara lain adalah waktu saat ini, parameter yang diambil dari hard disk sebagai entitas yang tidak dapat diprediksi, dan parameter-paramter lain sebagai entropi eksternal yang akan dibahas pada subbab selanjutnya.

Pada beberapa sistem operasi tanpa hard disk, seperti keluarga Linux Live dan keluarga Microsoft Windows PE, maka parameter yang dapat diperoleh hanyalah waktu saat ini. Parameter ini sangatlah lemah, mengingat bila seorang pengguna mengetahui waktu dengan ketepatan 1 detik, dan pewaktuan menyimpan waktu dengan ketepatan 1 micro detik, dan pewaktuan disimpan dalam 32-bit, maka hanya membutuhkan penyerangan dengan metode *brute force* sebanyak 60×10^6 serangan.

Dalam paper ini akan mengasumsikan bahwa sistem yang akan dibahas adalah sistem operasi yang memiliki sarana penyimpanan tetap (*permanent storage*). Pada sistem ini, biji acak (*random seed*) yang sedang berlaku akan disimpan pada saat sistem nonaktif, dan akan diambil kembali pada saat sistem aktif. Proses penyimpanan ini tentunya membutuhkan algoritma kriptografi simetrik yang berbeda-beda pada setiap sistem. Dibutuhkannya enkripsi pada file penyimpanan ini disebabkan file ini akan dapat dengan mudak diakses dan dimanipulasi pada saat sistem operasi sedang tidak aktif. Bila tanpa proses enkripsi, Pelaku kejahatan dapat dengan mudah memodifikasi isi file tersebut atau dengan mudah menimpa file tersebut dengan file sejenis lainnya.

3.3 Mengumpulkan entropi

Pada lingkungan PC, pembangkit bilangan acak akan mengumpulkan entropi, sebuah benda yang akan menentukan hasil angka acak yang dihasilkan. Entropi tersebut bersumber utama dari interupsi perangkat keras terhadap kernel sistem operasi. Interupsi tersebut dapat berupa perangkat keras masukan pengguna seperti: mouse, keyboard, diska, dll. Dapat juga berupa perangkat keras internal PC, seperti jam bios, cpu, memori, dll.

Setiap terdapat entropi, maka sistem akan merekamnya dalam 32 bit data (pada sistem 32 bit). 16 bit pertama merupakan informasi yang berkaitan dengan waktu entropi tersebut terekam dalam satuan mili detik semenjak sistem terboot, atau dalam satuan granularitas putaran cpu (*cpu cycles*). 16 bit berikutnya menandakan tipe entropinya dan nilai yang dikandungnya, sebagai contoh, bila user mengetikkan kunci pada keyboard, maka kunci yang diketikkan tersebut akan tersimpan pada 16 bit kedua tersebut. Untuk entropi-entropi lain akan sangat beragam tergantung sistem yang sedang berjalan.

Sumber-sumber masukan yang merupakan type-value pada prosedur penambahan entropi antara lain:

- Keyboard event. Type-value menyimpan kode dari penekanan dan pelepasan kunci dari keyboard. Range nilai yang valid adalah 0 hingga 255.
- Mouse event. Type-value := (type <<4) + code + (code >>4) + value
  Dengan type mendefiniskan type event yaitu menekan atau melepaskan tombol mouse, memulai pergerakan, atau mengakhiri pergerakan mouse. Code adalah tombol pouse yang bersesuaian (kiri, kanan, atau tengah) atau pemutaran roda tengah atau x dan y hasil dari proses pergerakan mouse. Value bernilai true/false pada kasus tombol mouse ditekan atau dilepaskan, 1 atau -1 untuk proses pemutaran roda, dan nilai perpindahan untuk proses pemindahan mouse. Secara singkat, data mouse adalah 32-bit dengan ukuran perpindahan sebagai faktor entropi utama. Namun, hanya 10 bit yang digunakan untuk perpindahan, 2 bit lainnya digunakan untuk tombol, sehingga faktanya adalah 12 dari 32 bit yang dipakai secara efektif.

- Disk event. Dihitung pada saat keberhasilan operasi terhadap disk (Ide, SCSI, Floppy, dll). Type-value nya dibangkitkan berdasarkan angka major dan minor pada sistem operasi Linux yang mensimbolkan sebuah angka unik untuk masing-masing perangkat.
  type-value := 0x100 + ((major << 20) | minor
  Jika hanya terdapat satu buah disk IDE, type-valuenya akan bernilai tetap, dikarenakan angka major dan minornya konstan. Umumnya angka major adalah 3 (IDE disk pertama) dan minor adalah 0 (master), dan type-value yang berhasil dikombinasikan adalah 0x200100. Diasumsikan sebuah mesin rata-rata tidak memiliki lebih dari 8 buah disk, maka range dari nilai type-value dilimatasi sepanjang 3 bit (2^3 bit = 8).
- Interrupt event. Hasil dari proses interrupt adalah nomor IRQ (*interrupt request channel*), dengan nomor yang valid dari 0 hingga 15. Dikarenakan keterbatasan ini, banyak sistem operasi yang mengabaikan entropi dari interrupt ini.

Pada lingkungan sistem lainnya, pembangkit bilangan acak tidak menggunakan perangkat keras masukan seperti layaknya PC, namun menggunakan sumber daya yang apa adanya. Sebagai contoh, sebuah OpenWRT router tidak memiliki hard disk, mouse, dan keyboard, sehingga perangkat tersebut tidak dapat menjadi sumber entropi. Namun router tersebut mengumpulkan entropi dari data-data yang melewati router tersebut.

3.4 Mengestimasi ukuran entropi

Salah satu bahasan fundamental dalam penggunaan entropi fisik adalah dalam mengestimasi besarnya entropi yang akan disuplai kepada pembangkit bilangan acak, dan besarnya entropi yang dapat dan sedang disimpan pada penampung entropi (*entropy pool*).

Seperti yang telah dijelaskan sebelumnya, bahwa pembangkit bilangan acak dapat terdiri dari dua buah, yaitu blocking dan non blocking. Pada pembangkit bilangan acak blocking, antar muka pembangkit bilangan acak tidak mengembalikan angka melebihi estimasi entropi saat ini sehingga memungkinkan terjadinya blocking. Sebaliknya pada pembangkit bilangan acak non blocking, akan mengembalikan sebarang angka ang dimilikinya. Perbedaan ini mempengaruhi estimasi entropi, dan mempengaruhi proses pembangkitan bilangan acak secara keseluruhan.

Pembangkit bilangan acak mengestimasi ukuran entropi sebuah kejadian sebagai fungsi pewaktuan saja, tidak tipe kejadian tersebut. Proses estimasi tersebut dilakukan dengan proses sebagai berikut:

Diketahui t_n merupakan pewaktuan kejadian (*event*) nomor n. Didefiniskan:

$\delta_n = t_n - t_{n-1}$

$\delta^2_n = \delta_n - \delta_{n-1}$

$\delta^3_n = \delta^2_n - \delta^2_{n-1}$

t_n, δ_n, δ^2_n, δ^3_n, masing-masing memiliki panjang 32 bit.

Ukuran dari entropi yang ditambahkan dari event didefiniskan sebagai log2 (min ($|\delta_n|$, $|\delta^2_n|$, $|\delta^3_n|$)). Jika hasilnya adalah 0, maka estimasinya berukuran 0. Walaupun estimasi berukuran 0, event tersebut tetap digunakan untuk mengupdate status pembangkit bilangan acak. Namun penghitungan entropi sendiri baru dilakukan bila estimasi entropi bernilai positif.

Estimasi ini hanya relevan pada kasus penambahan entropi dari sistem operasi ke pool. Bila user mengupdate manual, penghitung entropi tidak berubah. Bila n bit terekstraksi dari pool, estimasi entropi dikurangi sebesar n. Bila bit dipindahkan dari sebuah pool ke pool lainnya, maka yang dilakukan adalah mengurangi pool yang satu kemudian menambahkannya ke pool yang lain.

3.5 Memperbarui pool

Mekanisme pembaruan pool pembangkitan bilangan acak didasarkan pada TGFSR (Twisted Generalized Feedback Shift Register [1] dan [2]). Keuntungan utama dari TGFSR adalah panjang extended cycle dari initial seed. Waktu dari TGFSR dengan state sepanjang 128 word (pada PC 32-bit) dapat sepanjang $2^{128x32} - 1$ langkah.

Masukan satu-satunya dari TGFSR adalah nilai insial dari state (yang berupa seed sepanjang p x w bit), dan disetiap iterasi,

state internal digunakan untuk membangkitkan state yang baru.

Entropi ditambahkan pada setiap ronde dan menghasilkan hasil komputasi, juga pada saat entropi ditambahkan dari sumber eksternal, atau dari pool primer ke pool sekunder.

Setiap pool diperbarui berdasarkan perhitungan polynomial primitif. Bilangan polinomial diambil berdasarkan besarnya pool, sehingga pool sekunder memiliki rumur polinomial $x^{32} + x^{26} + x^{20} + x^{14} + x^7 + x + 1$. Sedangkan pool primer memiliki rumus $x^{128} + x^{103} + x^{76} + x^{51} + x^{25} + x + 1$, penambahan entropi pada kedua pool tersebut identik dengan pool-pool yang lebih kecil, kecuali menggunakan polynomial untuk memperbarui pool.

Penambahan entropi dapat dianalisis dengan mengasumsikan pembangkit bilangan acak akan menghitung ulang seed pada setiap iterasinya. Proses alternatif adalah, sebuah pool dapat menganalisa dengan mengasumsikan bahwa TGFSR digunakan untuk mengenkripsi masukan entropi. Proses penghitungan ulang seed pada pembangkitan bilangan acak semu mengubah properti dari dasar TGFSR. Periode yang lama tidak lagi dapat dipastikan, dan proses tidak lagi berupa fungsi linear dari state initial.

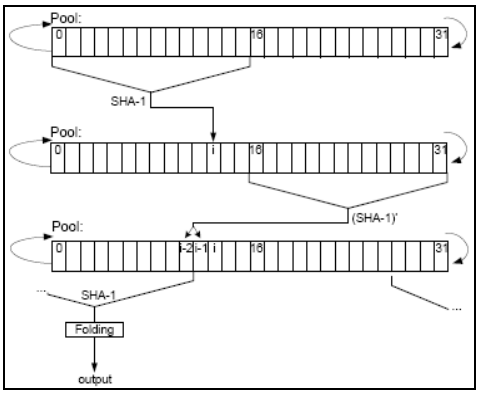
```
Algoritma Penambahan(pool, j, g):
temp := g
temp := temp xor pool[j]
temp := temp xor pool[(j+1) mod 32]
temp := temp xor pool[(j+7) mod 32]
temp := temp xor pool[(j+14) mod 32]
temp := temp xor pool[(j+20) mod 32]
temp := temp xor pool[(j+26) mod 32]
temp := (temp >> 3) xor tabel[temp & 7]
// 3 bit terakhir dari temp memilih
// sebuah tabel masukan yang ter xor
// dengan (temp >> 3)
pool[j] := temp
// tabel[] didefinisikan sebagai
// berikut:
// tabel[0] = 0x0
// tabel[1] = 0x3b6e20c8
// tabel[2] = 0x76dc4190
// tabel[3] = 0x4db26158
// tabel[4] = 0xedb88320
// tabel[5] = 0xd6d6a3e8
// tabel[6] = 0x9b64c2b0
// tabel[7] = 0xa00ae278
```

3.6 Mengekstraksi bit-bit acak

Entropi yang berhasil dikumpulkan dan disimpan pada pool-pool kemudian akan di ekstraksi manakala seorang user membutuhkan sebuah angka acak. Proses ekstraksi dilakukan pada semua pool.

Mengekstraksi sebuah entropi dari sebuah pool tidaklah merupakan operasi yang sederhana. Operasi ini melibatkan proses hashing bit-bit yang terekstraksi, mengubah state dari sebuah pool, kemudian menguarangi estimasi entropi dengan angka hasil bit-bit yang terekstraksi.

```
Algoritma Ekstraksi(pool, nbytes, j):
while nbytes > 0
    tmp := SHA-1(pool[0..15])
    //Hasilnya merupakan 5 x 16 bit
    add(pool, j, tmp[0])
    tmp := SHA-1'(pool[16..31])
    add(pool, j-1 mod 32, tmp[2])
    add(pool, j-2 mod 32, tmp[4])
    tmp := SHA-1'(pool[(j-2-15) mod 32
        ...(j-2) mod 32])
    tmp := folding(tmp[0..4])
    //Hasilnya merupakan 2.5 x 16 bit
    output(tmp, min(nbytes, 10))
    nbytes := nbytes-min(nbytes, 10)
    j := j-3 mod 32
end while
```



Kedua gambar diatas memperlihatkan pseudo-code dan sebuah diagram algoritma ekstraksi entropi. Proses ekstraksi entropi dilakukan menggunakan blok-blok berukuran 10 byte. Untuk penyederhanaan deskripsi diatas tidak mengikutkan langkah-langkah pengurangan estimasi entropi dan proses pengisian kembali entropi. Algoritma tersebut memproses fungsi SHA-1 ke 16x16 bit pertama, dan menambahkan hasilnya ke lokasi j. Kemudian dia mengaplikasikan sebuah varian SHA-1, yang kita sebut dengan SHA-1' kepada separo kedua dari pool, dan menambahkan hasilnya ke lokasi j-1 dan j-2. Yang terakhir, dia mengaplikasikan SHA-1' ke 16x16 bit yang berakhir di lokasi j-2, dan menggunakan hasilnya untuk mengkomputasikan keluaran dengan cara berikut ini: Keluaran dari SHA-1 adalahan sepanjang 20 byte. Keluran ini

akan di lipat, dan menghasilkan 10 byte yang merupakan keluaran dari iterasi. Keluaran ini kemudian disalin ke buffer target (user maupun kernel), dan angka dalam byte-byte yang ingin disalin telah diperbarui. Loop tersebut terus berlangsung hingga angka yang diminta berhasil dikeluarkan.

Fungsi hash yang digunakan terhadap separo byte pertama merupakan fungsi SHA-1 asli, lihat [3] untuk penjelasan lebih lanjut tentang SHA-1 yang asli. Namun setiap operasi hash berikutnya, yang disebut dengan SHA-1', menggunakan konstanta 5 buah intial value (IV) dari 5 buah keluaran dari fungsi hash sebelumnya.

Bit-bit terekstraksi dari pool primer manakala dibutuhkan untuk menghitung ulang pool-pool yang lain. Pada kasus ini, algoritmanya agak sedikit berbeda dikarenakan pool primer lebih panjang. Fungsi operasi SHA-1 (atau SHA-1') diaplikasikan ke setiap dari 16x16 bit dari pool primer, dan sekali lagi ke 16x16 bit yang berakhir di lokasi j-8. Delapan buah operasi SHA-1 memperbarui delapan 16 bit pada pool, dan operasi yang terakhit menghasilkan keluaran yang diminta.

```
Algoritma Ekstraksi(pool, nbytes, j):
while nbytes > 0
    tmp := SHA-1(pool[0..15])
    //Hasilnya merupakan 5 x 16 bit
    add(pool, j, tmp[0])
    tmp := SHA-1'(pool[16..31])
    add(pool, j-1 mod 128, tmp[2])
    tmp := SHA-1'(pool[32..47]
    add(pool, j-2 mod 128, tmp[4])
    tmp := SHA-1'(pool[48..63]
    add(pool, j-3 mod 128, tmp[1])
    tmp := SHA-1'(pool[64..79]
    add(pool, j-4 mod 128, tmp[3])
    tmp := SHA-1'(pool[80..95]
    add(pool, j-5 mod 128, tmp[0])
    tmp := SHA-1'(pool[96..111]
    add(pool, j-6 mod 128, tmp[2])
    tmp := SHA-1'(pool[112..127]
    add(pool, j-7 mod 128, tmp[4])
    add(pool, j-8 mod 128, tmp[1])
    tmp := SHA-1'(pool[(j-8) mod 128
        ...(j-8-15) mod 128])
    tmp := folding(tmp[0..4])
    //Hasilnya merupakan 2.5 x 16 bit
    output(tmp, min(nbytes, 10))
    nbytes := nbytes-min(nbytes, 10)
    j := j-9 mod 32
end while
```

4. Algoritma Pembangkit Bilangan Acak Semu

Terdapat beberapa algoritma yang sering dan umum dipakai di bidang kriuptografi untuk membangkitkan bilangana acak semu. Algoritma tersebut adalah: LCG (linear Congruential Generator) dan BBS (Blum-Blum Shub). Selain kedua algoritma diatas masih banyak algoritma lain, semisal: algoritma Mersenne Twister, algoritma Knuth, dll.

Algoritma pembangkit bilangan acak adalah algoritma dengan masukan minimum dapat menghasilkan suatu deret bilangan acak. Bila masukannya sama, maka akan menghasilkan urutan bilangan acak yang sama pula.

4.1 Syarat Algoritma Pembangkit Bilangan Acak

Algoritma pembangkit bilangan acak harus setidaknya memenuhi syarat-syarat yang telah ditentukan pada Request For Commands 1750 [10] (Randomness Recommendations for Security). Ringkasan dari RFC 1750 adalah sebagai berikut:

- Tidak dapat diprediksi.
  Suatu algoritma pembangkit bilangan acak harus bersifat tidak dapat diprediksi hasil keluarannya. Sebab dengan dapat memprediksi keluaran suatu algoritma, maka dengan tidak langsung dapat pula mengetahui kelauran dari fungsi kriptografi secara keseluruhan.
- Untuk algoritma yang berdasarkan pada perangkat keras: Tahan akan skew (pembelokan). Yaitu bila perangkat keras dimanipulasi, maka algoritma dapat tahan terhadap manipulasi sumber bilangan tersebut hingga level tertentu yang tidak mungkin lagi dapat ditahan.
- Untuk algoritma yang berdasarkan perangkat lunak/aritmatika: Memiliki fungsi penggabungan dan pengacakan. Yang berguna untuk mencegah penyerangan terhadap algoritma secara keseluruhan.

4.2 Linear Congruential Generator (LCG)

Linear Congruential Generator adalah pembangkit bilangan acak tertua yang sangat terkenal. Seperti halnya algoritma pembangkit bilangana acak lainnya, LCG

membutuhkan masukan berupa biji/umpan (seed) sebagai masukan utamanya. Formula LCG dalam relasi rekursif dan penjelasannya didefinisikan sebagai berikut [11]:

$x_n = (ax_{n-1} + b) \bmod m$

Keterangan:

x_n = bilangan acak ke-n dari deretnya

x_0 = umpan/biji (seed)

x_{n-1} = bilangan acak sebelumnya

a = faktor pengali (konstan)

b = increment (konstan)

m = modulus (konstan)

LCG memiliki periode tidak lebih besar dari m, dan pada kebanyakan kasus periodenya kurang dari itu. LCG mempunyai periode penuh (m-1) jika memenuhi syarat berikut:

- b relatif prima terhadap m
- a-1 dapat dibagi dengan semua faktor prima dari m
- a-1 adalah kelipatan 4 jika m adalah kelipatan 4
- a>0 dan b>0

Secara teori, LCG mampu menghasilkan urutan bilangan acak yang baik, namun hasil tersebut tergantung dari pemilihan konstanta a,b, dan m. Pemilihan nilai yang buruk dapat menyebabkan hasil algoritma menghasilkan urutan bilangan acak yang kurang baik. Dan karena sifatnya yang dapat diprediksi, maka algoritma kriptografi ini kurang cocok untuk dipakai dalam bidang kriptografi.

Keunggulan dari algoritma ini adalah kecepatannya yang baik, dikarenakan operasi yang dilakukan hanyalah beberapa operasi manipulasi bit saja.

4.3 Blum-Blum Shub (BBS)

Blum-blum shub cukup terkenal sebagai sebuah algoritma pembangkit bilangan acak yang baik dikarenakan kemangkusannya dan kesederhanaannya. BBS dibuat pada tahun 1986 oleh tiga orang: Lenore Blum, Manuel Blum, dan Michael Shub. Algoritma/langkah-langkah pembangkitan bilangan acak BBS adalah sebagai berikut [11]:

- Pilih dua buah bilangan prima rahasia, p dan q. Yang keduanya kongruen dengan 3 modulo 4.
- Kalikan keduanya sehingga menghasilkan n. Bilangan ini disebut bilangan bulat Blum.
- Pilih bilangan bulat acak lain, s, sebagai umpan sedemikian hingga:
  1. $2 <= s < n$
  2. s dan n relatif prima

  Kemudian hitung $x_0 = s^2 \bmod n$
- Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:
  1. Hitung $x_i = x_{i-1}^2 \bmod n$
  2. z_i = bit LSB dari x_i

Yang menarik dari algoritma pembangkit bilangan acak ini adalah bahwa tidak perlu melakukan iterasi untuk mendapatkan bilangan acak jika p dan q diketahui, sebab xi dapat dihitung secara langsung.

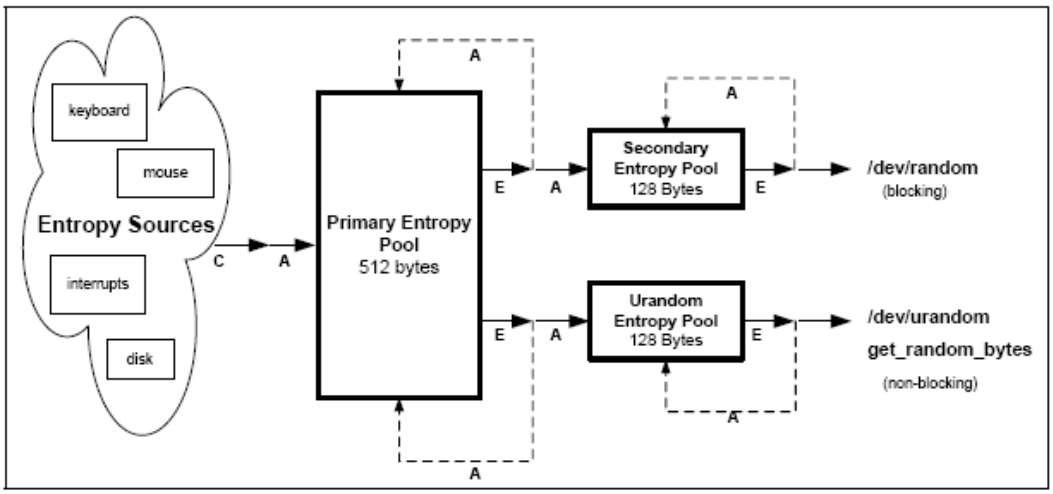
Keamanan BBS terletak pada sulitnya memfaktorkan n. Nilai n tidak perlu rahasia dan dapat diumumkan kepada publik. BBS tidak dapat diprediksi dari arah kiri dan tidak dapat diprediksi dari arah kanan, yaitu berarti jika diberikan barisan bit yang dihasilkan oleh BBS, kriptanalis tidak dapat menganalisa barisan bit sebelumnya dan barisan bit sesudahnya dalam urutan bilangan acak yang dihasilkan.

5. Pembangkitan Bilangan Acak Semu pada Sistem Operasi Linux

Kernel Linux merupakan sebuah proyek open source yang telah dikembangkan mulai lebih dari 15 tahun yang lalu dipimpin oleh Linus Torvalds. Kernel linux merupakan bagian inti dari linux yang mengandung fungsi-fungsi utama sistem operasi Linux. Kernel linux merupakan bagian tak terpisahkan yang merupakan satu-satunya bagian linux yang dijamin sama untuk semua distribusi Linux.

Kernel linux seperti dijelaskan diatas merupakan bagian inti yang mendukung keseluruhan sistem operasi untuk bekerja. Salah satunya adalah dengan mendukung sarana-sarana agar program-program lain dapat bekerja.

Berkaitan dengan pembangkit bilangan acak, Kernel Linux menyediakan sarana tersebut



dengan baik. Dengan demikian program-program lain dapat menggunakannya tanpa perlu membuat sendiri pembangkit bilangan acak.

5.1 Device driver pembangkit bilangan acak

Sistem operasi linux menganggap semua hal sebagai sebuah file. Hal tersebut dapat berupa file sendiri hingga perangkat keras. Sehingga dalam membaca event yang diberikan perangkat keras, semisal: mouse, cukup hanya dengan membaca file device driver mouse itu sendiri.

Sebaliknya demikian pula, sebuah file dapat dianggap pula sebagai sebuah perangkat keras virtual. Yaitu dengan mengimplementasikan sebuah device driver yang disebut character device driver.

Dalam kasus pembangkitan bilangan acak, sistem operasi linux menyediakan dua buah file yaitu /dev/random dan /dev/urandom. Kedua file tersebut merupakan sebuah perangkat keras virtual. Yaitu seakan-akan terdapat sebuah perangkat keras yang terhubung ke sistem yang bertugas menyuplai bilangan acak.

Dengan membaca kedua file tersebut maka user dapat memperoleh bilangan acak. Membaca dapat dengan membuka file tersebut pada baris-baris program, atau benar-benar membuka file tersebut di shell linux.

Perbedaan kedua file tersebut adalah jika /dev/random merupakan pembangkit bilangan acak yang bersifat blocking, sedangkan /dev/urandom tidak bersifat blocking. Pembahasan tentang ini telah dibahas pada subbab 3.1.

Seperti dibahas pada subbab 3.1, pembangkit yang bersifat blocking berkinerja lebih buruk daripada pembangkit yang bersifat non blocking, dikarenakan setiap user dalam meminta suatu bilangan acak, maka harus menunggu perhitungan terlebih dahulu. Namun di satu sisi yang lain, pembangkit yang bersifat blocking bersifat sangat aman dibandingkan pembangkit bersifat non blocking.

5.2 Application Programming Interface pembangkit bilangan acak pada kernel space

Program-program yang berjalan pada sistem operasi linux dibagi menjadi dua kelompok. Kelompok pertama adalah program-program yang berjalan diatas kernel space dan kelompok kedua adalah program-program yang berjalan diatas user space.

Program yang berjalan diatas kernel space merupakan program yang bersifat penting bagi berlangsungnya sistem. Kelebihannya adalah lebih efektif dalam pemakaian sumber daya sistem, sedangkan kekurangannya adalah dapat menimbulkan kegagalan sistem bila programnya terdapat kekurangan.

Application Programming Interface adalah sebuah antar muka bagi seorang programmer dalam memanfaatkan fungsi-fungsi pustaka yang telah disediakan oleh suatu sistem atau programmer lain.

Kernel linux menyediakan sebuah API pada kernel space untuk memudahkan programmer dalam memanfaatkan fungsionalitas pembangkit bilangana acak. Sebuah file header yang memberikan informasi lengkap tentang pembangkit bilangan acak di user space adalah pada file <linux/random.h>. Antar muka tersebut merupakan sebuah fungsi bernama get_random_bytes dan get_random_int. Deskripsi fungsi tersebut adalah:

```
#include <linux/random.h>

extern void get_random_bytes(void* buf,
int nbytes);

Parameter ke-1 (buf):
Alamat memori tempat dimana API akan
mengisi dengan angka acak.

Parameter ke-2 (nbytes):
Besarnya ukuran alamat memori tersebut
untuk dipenuhi dengan angka acak.
```

```
#include <linux/random.h>

unsigned int get_random_int(void);

Mengembalikan:
Sebuah bilangan acak dengan range nilai
antara 0 hingga 65535
```

5.3 Application Programming Interface pembangkit bilangan acak pada user space

Sistem operasi linux selain menyediakan API pembangkit bilangan acak untuk program-program yang berjalan diatas kernel space juga menyediakan API pembangkit bilangan acak untuk program-program yang berjalan diatas user space. Hal ini dikarenakan program yang berjalan diatas user space tidak dapat memanfaatkan API yang ditujukan khusus untuk program yang berjalan diatas kernel space, begitu pula sebaliknya.

File header yang berisi lengkap tentang fungsionalitas pembangkit bilangan acak pada user space adalah file <stdlib.h>. Terdapat tiga fungsi utama pembangkit bilangan acak, yaitu: rand, rand_r, dan srand. Dan terdapat sebuah konstanta bernama MAX_RAND yang sangat berguna.

```
#include <linux/random.h>

#define MAX_RAND ...

Konstanta MAX_RAND sangat berguna untuk
menentukan range nilai yang dapat
dikeluarkan oleh fungsi-fungsi lain.
Konstanta tersebut akan berbeda-beda di
setiap sistem, sehingga lebih mudah
jika hanya menyebut dengan nama
MAX_RAND saja.
```

```
#include <stdlib.h>

extern int rand(void) __THROW;

Mengembalikan:
Sebuah bilangan acak antara 0 hingga
MAX_RAND.
```

```
#include <linux/random.h>

extern void srand(unsigned int __seed)
__THROW;

Parameter ke-1 (__seed):
Seed baru sebagai pembangkit bilangan
acak fungsi rand(). Untuk membangkitkan
sebuah urutan bilangan acak yang sama,
maka dengan menset seed dengan angka
yang sama dapat membangkitkan sekuen
angka acak yang sama.
```

```
#include <linux/random.h>

extern int rand_r(unsigned int *__seed)
__THROW;

Parameter ke-1 (__seed):
Angka seed, yang juga merupakan
parameter output sebgai penanda status.
Dapat dipakai pada program yang
memiliki multi thread.

Mengembalikan:
Sebuah bilangan acak dengan range nilai
antara 0 hingga MAX_RAND.
```

6. Pembangkitan Bilangan Acak Semu pada Sistem Operasi Windows

Pada sistem operasi Microsoft Windows agak sedikit berbeda dibandingkan pada sistem operasi Linux. Tiga tahap utama dalam pembangkitan bilangan acak tetaplah sama, dari pengambilan entropi yang berasal dari perangkat keras eksternal, pengumpulan entropi ke dalam pool-pool yang disediakan, hingga pemberian entropi ke user.

6.1 Application Programming Interface pembangkit bilangan acak

Perbedaan antara sistem operasi Microsoft Windows dengan Linux adalah pada API di level kernel. Dikarenakan pada sistem operasi Microsoft Windows, program tidak dibedakan antara kernel space dengan user

space, namun keseluruhan program yang berjalan dianggap berjalan di user space.

Oleh karena itu, API pembangkit bilangan acak yang disediakan adalah API untuk user space, tidak ada API khusus kernel space. API di tingkat user space pun sangat mirip dengan API pembangkit bilangan acak di tingkat user space pada sistem operasi Linux. Hal ini dikarenakan kedua sistem operasi berlandaskan pada bahasa C. Sehingga API-APInya mengikuti kaidah standar C yang telah dirumuskan bersama.

```
#include <linux/random.h>

#define MAX_RAND ...

Konstanta MAX_RAND sangat berguna untuk
menentukan range nilai yang dapat
dikeluarkan oleh fungsi-fungsi lain.
Konstanta tersebut akan berbeda-beda di
setiap sistem, sehingga lebih mudah
jika hanya menyebut dengan nama
MAX_RAND saja.
```

```
#include <stdlib.h>

extern int rand(void) __THROW;

Mengembalikan:
Sebuah bilangan acak antara 0 hingga
MAX_RAND.
```

```
#include <linux/random.h>

extern void srand(unsigned int __seed)
__THROW;

Parameter ke-1 (__seed):
Seed baru sebagai pembangkit bilangan
acak fungsi rand(). Untuk membangkitkan
sebuah urutan bilangan acak yang sama,
maka dengan menset seed dengan angka
yang sama dapat membangkitkan sekuen
angka acak yang sama.
```

```
#include <linux/random.h>

extern int rand_r(unsigned int *__seed)
__THROW;

Parameter ke-1 (__seed):
Angka seed, yang juga merupakan
parameter output sebgai penanda status.
Dapat dipakai pada program yang
memiliki multi thread.

Mengembalikan:
Sebuah bilangan acak dengan range nilai
antara 0 hingga MAX_RAND.
```

6.2 API khusus bernama CryptoAPI

Pada sistem operasi Windows, Microsoft menyediakan sebuah pustaka khusus untuk masalah kriptografi, dikenal dengan CryptoAPI. Semua permasalahan kriptografi dapat dipecahkan dengan menggunakan pustaka ini. Dalam pustaka ini juga tersedia pembangkit bilangan acak. Pembangkit bilangan acak ini di klaim lebih baik daripada pembangkit bilangan acak di WINAPI biasa.

Pada pustaka ini juga dapat berhubungan dengan perangkat keras pembangkit bilangan acak khusus yang terhubung ke sistem. Banyak perangkat keras khusus pembangkit bilangan acak yang memantau kondisi fisik sekitar sebagai penentu bilangan acak, semisal radioaktif atom, noise suara, gelombang elektromagnetik, panas CPU, dll. Namun alat-alat tersebut tidak umu ada di masyarakat, kecuali panas CPU. Keluarga Intel Pentium III keatas telah menyediakan failitas tersebut (yang akan lebih dijelaskan pada bab selanjutnya). API CryptoAPI mampu menangkap bilangan acak tersebut dan dipakai pada program-program yang berjalan.

7. Pembangkitan Bilangan Acak pada perangkat CPU

Selain berasal dari perangkat lunak seperti pada bab-bab sebelumnya, bilangan acak juga dapat didapatkan dari perangkat keras. Perangkat keras yang telah memiliki fungsi pembangkit bilangan acak salah satunya adalah beberapa tipe prosesor. Selain prosesor ada peralatan lain yang benar-benar berfungsi untuk membangkitkan bilangan acak dari sumber-sumber eksternal semisal radioaktif atom, noise gelombang, dll, namun alat tersebut jarang ditemui dan memang dirancang untuk kebutuhan tertentu.

Mulai dari keluarga Pentium III [9] keatas (dengan chipset nomor 82802 Firmware Hub), intel telah menanamkan fungsionalitas pembangkit bilangan acak pada firmware prosesornya. Fungsionalitas tersebut ditanamkan sebagai salah satu perintah interrupt yang dapat diberikan kepada prosesor. Untuk dapat memanggil perintah tersebut dapat dengan membuat beberapa baris program assembly.

7.1 Sumber pembangkit bilangan acak pada CPU

Sumber entropi sebagai penentu bilangan acak pada CPU adalah noise temperatur pada resistor yang terdapat di cpu. CPU merupakan bagian di komputer yang suhunya selalu berubah dengan cepat, dan dengan teknologi saat ini cukup mudah untuk mengetahui suhu CPU hingga ke suatu ukuran presisi. Dengan demikian suhu pada CPU sangat baik untuk dijadikan sumber pembangkit bilangan acak.

Setelah membaca sumber entropi tersebut langkah selanjutnya sebelum diberikan kepada user adalah dengan memasukkannya ke dalam fungsi hash SHA-1. Setelah terhash, hasilnya itulah yang akan diberikan kepada user.

Namun dikarenakan suhu merupakan satu-satunya sumber entropi, maka dari segi keamanan dapat diaktakan kurang aman. Sebab dengan demikian penyerang dapat dengan mudah memodifikasi satu-satunya sumber entropi tersebut. Salah satu caranya adalah dengan sengaja membuat CPU bersuhu sesuai dengan yang dikehendakinya dengan bantuan benda lain, semisal es, dll. Atau dengan memodifikasi sensor pembaca suhu shingga salah dalam membaca suhu tersebut.

7.2 Penggunaan bilangan acak pada prosesor

Untuk menggunakan bilangan acak pada prosesor perlu suatu mekanisme pemanggilan fungsi yang terdapat pada prosesor tersebut. Untuk kasus prosesor Intel, Intel telah menawarkan sebuah perangkat lunak driver yang dapat diunduh. Driver tersebut dapat menghubungi pembangkit bilangan acak yang terdapat di firmware prosesor, dan dapat memberikannya ke aplikasi lain.

Pada Microsoft Windows 95 keatas, telah tersedia pula sebuah CrptoAPI yang terdapat pada paket WINAPI. CryptoAPI tersebut telah built in memiliki fungsionalitas dapat mengecek ada tidaknya fungsionalits pembangkit bilangan acak pada prosesor, dan bila ada, dapat menggunakannya langsung.

8. Pembangkitan Bilangan Acak Semu pada pustaka-pustaka umum

8.1 Pustaka .Net Framework 2.0

Pada pustaka .Net Framework 2.0 pembangkit bilangan acak dibagi menjadi dua, pembangkit bilangan acak yang dibangkitkan menggunakan algoritma dari D.E. Knuth [8], dan algoritma yang diberikan oleh Cryptographic Serviece Profider dari .net framework.

Yang pertama dikarenakan menggunakan algoritma yang telah umum dipakai di masyarakat sehingga tingkat keamanannya tidak mencukupi untuk permasalahan kriptografi yang ada. Namun cukup untuk sekedar meminta bilangan acak semu. API ini dapat menerima seed seperti halnya API-API pembangkit bilangan acak yang lain. Pembangkit bilangan acak ini diserahkan kepada sebuah kelas System.Random yang menanganui seluruh permintaan bilangan acak.

```
Namespace: System
Assembly: mscorlib (mscorlib.dll)
public class Random
{
        int Next();
        void NextBytes(byte[] buffer);
        double Nextdouble();
}
```

Yang kedua, pembangkit bilangan acak yang disediakan oleh Cryptographic Serviece Provider menyediakan pembangkit bilangan acak yang lebih *reliable*. Dikarenakan algoritma pembangkitannya tidak dipublikasikan. Pembangkita bilangan acak ini cukup dapat diandalkan mengingat seluruh fungsionalitas kriptografi pada .net framework 2.0 bergantung sepenuhnya pada pembangkit bilangan acak ini saat membutuhkan bilangan acak.

```
Namespace: System.Security.Cryptography
Assembly: mscorlib (mscorlib.dll)
public class RNGCryptoServiceProvider
{
        void GetBytes(byte[] data);
}
```

8.2 Java 2 Second Edition 1.5

Seperti halnya .net framework 2.0, j2se 1.5 juga menyediakan dua buah kelas pembangkit bilangan acak. Kelas pertama

merupakan pembangkit bilangan acak yang bersifat secukupnya, kurang baik dari segi keamanan, namun cukup baik untuk kebutuhan akan bilangan acak sederhana. Kelas tersebut juga seperti halnya di .net framework 2.0, merupakan implementasi dari algoritma yang dikembangkan oleh Knuth [8].

```
Namespace: java.util
public class Random
{
        void nextBytes(byte[] bytes);
        int nextInt();
        float nextFloat();
}
```

Kelas berikutnya adalah penegmbangan dari kelas pembangkit bilangan acak sebelumnya. Dalam kelas ini kekurangan dari segi keamanan diperbaiki, sehingga kelas ini dapat dipakai sebagai kelas pendukung fungsionalitas kriptografi pada framework java secara keseluruhan.

```
Namespace: java.security
public class SecureRandom
{
        Void nextBytes(byte[] bytes);
}
```

8.3 Pustaka PHP

Pada pustaka php, pembangkit bilangan random menggunakan native API pembangkit bilangan acak di sistem operasi yang berjalan. Bila di Microsoft Windows akan menggunakan WINAPI pembangkit bilangan acak, dan bila di Linux akan menggunakan API yang bersesuaian pula.

```
int rand();
Mengembalikan: bilangan acak dari 0 hingga MAX_RAND
```

Namun selain itu, terdapat pula sebuah fungsi pembangkit bilangan acak lain yang menggunakan prinsip kerja dari algoritma Mersenne Twister. Yang di klaim dapat menghasilkan bilangan acak sebaik native API namun dengan kecepatan pemrosesan empat kali lebih cepat.

```
int mt_rand();
Mengembalikan: bilangan acak dari 0 hingga MAX_RAND namun dengan menggunakan implementasi algoritma Mersenner Twister
```

9. Kesimpulan

Dalam dunia kriptografi, pembangkitan bilangan acak merupakan suatu bagian yang tidak dapat dipisahkan. Bilangan acak dibutuhkan untuk menghasilkan sebuah password acak, untuk menghasilkan cipher text yang baik, untuk menghasilkan kunci yang baik, dll.

Pembangkit bilangan acak dapat ditemui pada setiap sistem operasi yang telah ditanamkan pada kernelnya. Sistem operasi yang dibahas adalah Linux dan Mincrosoft Windows. Keduanya memiliki kemampuan pembangkitan bilangan acak secara langsung dari kernel. Keduanya memiliki opsi pula, satu adalah pembangkit bilangana cak yang sangat aman, namun dalam hal kecepatan sangat lambat, dan yang lainnya adalah sangat cepat, tapi dari segi keamanan kurang baik.

Bilangan acak dihasilkan dari tiga tahap utama. Tahap pertama adalah mengampil entropi atau data eksternal yang tidak dapat diprediksi. Entropi seperti: pergerakan mouse, kunci yang terketik pada keyboard, data dari jaringan, dll. Entropi tersebut merupakan sumber utama pembangkit bilangan acak.

Tahap kedua adalah mengumpulkannya dan menerapkan algoritma pemilihan bilangan acak. Dalam proses mengumpulkan entropi ke dalam pool bukanlah hal yang mudah, sebab terdapat banyak pool entropi, dan tidak dapat dengan sebarang memasukkannya ke dalam pool. Banyak faktor yang harus dipenuhi untuk memasukkan entropi ke pool yang bersesuaian ditujukan agar bilangan yang dihasilkan benar-benar acak. Setelah dimasukkan ke dalam pool, kemudian dilakukan komputasi terhadap pool tersebut menggunakan beberapa pilihan algoritma pembangkitan bilangan acak, yang di setiap sistem dapat berbeda. Algoritma tersebut akan mengubah entropi mentah menjadi entropi siap jadi.

Langkah ketiga adalah memberikannya ke user. Entropi siap jadi yang dihasilkan dari proses sebelumnya akan diberikan kepada user. Sebelum diberikan dilakukan pemrosesan akhir, yaitu diterapkan proses

hashing denga funghsi hash yang umumnya adalah SHA-1.

Ketiga langkah tersebut tidaklah sebentar dalam hal pemrosesannya, untuk itu, pada opsi pembangkit yang aman, dibutuhkan waktu yang tidak singkat oleh user dalam meminta bilangan acak. Seperti dalam linux, bila user membaca file /dev/random akan membutuhkan waktu 5 detikan untuk mendapatkan angka acak tersebut. Namun untuk opsi yang lain yang kurang aman, API tersebut akan memberikan angka acak seadanya kepada user bergantung angka yang sedang ada di pool saat ini.

Diatas level tersebut terdapat API yang mengubungkan antara programmer dengan fungsionalitas tersebut. Di dalam sistem operasi Linux, terdapat API untuk program-program yang berjalan di atas kernel space dan program-program yang berjalan di atas user space. Kernel space berarti program yang dapat berjalan langsung diatas kernel, lebih efisien, namun lebih berbahaya. Sedangkan user space berarti program yang berjalan dibawah kendali suatu program lain, lebih tidak efisien, namun lebih aman.

Pada sistem operasi Windows, tidak dikenal kernel space dan user space, semua program berjalan diatas user space. Di sini juga terdapat API dalam paket WINAPI untuk membangkitkan bilangan acak.

Dari API tersebut juga terdapat dua macam pembangkit bilangan acak. Pembangkit pertama kurang efisien untuk dipakai sebab perlu waktu untuk menunggu proses pembuatan bilangan acak yang tidak singkat, namun dari segi keamanan dapat dijamin lebih baik. Sedangkan pembangkit kedua, walaupun dari segi kecepatan sangat cepat, tidak perlu menunggu, namun dari segi keamanan kurang baik.

API tersebut dapat langsung dipakai pada level programming, namun banyak framework-framework yang biasa dipakai oleh pengembang program memberikan pustaka pembangkit bilangan acaknya sendiri. Sebut saja .net framework 2.0, java 2 second edition, dll. Dibalik pustakanya tersebut ada bagian yang memakai API bawaan kernel, namun ada pula yang memakai algoritma sendiri dalam pembangkitan bilangana acaknya. Biasanya algoritma yang dipakai dalam pembangkitan bilangan acak adalah algoritma dari Knuth [8]. Dalam beberapa pustaka juga memakai algoritma pembangkit bilangan acak dari Mersenne Twister.

Selain berbentuk perangkat lunak ataupun algoritma, pembangkit bilangana acak juga tersedia pada perangkat keras. Perangkat keras yang sering ditemui di masyarakat dan telah menyediakan fungsionalitas pembangkit bilangan acak adalah prosesor. Mulai keluarga Pentium III keatas, Intel telah menanamkan pembangkit bilangan acak yang sumber bilangannya adalah suhu prosesor.

Suhu prosesor pada CPU sering berganti oleh karena itu dengan memasang sensor dengan ketepatan tertentu, sumber panas tersebut dapat menjadi pembangkit bilangan acak yang baik.

Untuk memanggil fungsionalitas tersebut, dapat dengan memanggil interrupt prosesor dengan bahasa assembly, atau dengan memanfaatkan driver yang telah disediakan oleh Intel, atau dengan memanfaatkan CryptoAPI pada keluarga Microsoft Windows 95 keatas.

Daftar Pustaka

[1] M. Matsumoto and Y. Kurita. Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation*, 2(3):179–194, 1992.

[2] M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation*, 4(3):254–266, 1994.

[3] D. Eastlake 3rd, and P. Jones. US secure hash algorithm 1 (SHA1). Request for Comments 3174, Internet Engineering Task Force, September 2001.

[4] I. Goldberg and D. Wagner. Randomness and the Netscape browser. Dr Dobb's, pages 66–70, January 1996.

[5] Z. Gutterman and D. Malkhi. Hold your sessions: An attack on Java session-id generation. In A. J. Menezes, editor, CT-RSA, LNCS vol. 3376, pages 44–57. Springer, February 2005.

[6] B. Barak and S. Halevi. A model and architecture for pseudo-

random generation with applications to /dev/random. In ACM Conference on Computer and Communications Security, pages 203–212, 2005.

[7] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic attacks on pseudorandom number generators. In Fast Software Encryption, LNCS vol. 1372, pages 168–188. Springer, 1998.

[8] D. E. Knuth. "The Art of Computer Programming, volume 2: Seminumerical Algorithms". Addision-Wesley, Reading, MA, second edition, 1981.

[9] Intel® Security Driver. http://www.intel.com/design/software/drivers/platform/security.htm. Tanggal akses: 1 Januari 2007.

[10] Request For Commands 1750. http://www.ietf.org/rfc/rfc1750.txt. Tanggal akses: 1 Januari 2007.

[11] Munir, Rinaldi. Diktat Kuliah IF5054 Kriptografi. Institut Teknologi Bandung. 2006.

[12] Vicaria, Fernando. The Random Facts of Coding (or so it seems). http://blogs.msdn.com/fvicaria/archive/2004/10/06/238862.aspx. Tanggal akses: 1 Januari 2007.

[13] Callas, Jon. Using and Creating Cryptographic-Quality Random Numbers. http://www.merrymeet.com/jon/usingrandom.html. Tanggal akses: 1 Januari 2007.

[14] Coleridge, Robert. The Cryptography API, or How to Keep a Secret. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncapi/html/msdn_cryptapi.asp. Tanggal akses: 1 Januari 2007.

[15] Marienne Twister. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. Tanggal akses: 1 Januari 2007.

[16] Holes in the Linux random number generator?. http://lwn.net/Articles/184925/. Tanggal akses: 1 Januari 2007.

[17]. Gutterman, Zvi. Analysis of the Linux Random Number Generator. http://eprint.iacr.org/2006/086. Tanggal akses: 1 Januari 2007.

[18] Howard, Michael. Cryptographically Secure Random number on Windows without using CryptoAPI. http://blogs.msdn.com/michael_howard/archive/2005/01/14/353379.aspx. Tanggal akses: 1 Januari 2007.

[19] Wikipedia: Random number generator. http://en.wikipedia.org/wiki/Random_number_generator. Tanggal akses: 1 Januari 2007.