Studi Pemanfaatan Mersenne Twister sebagai *Secure Random Number Generator* dan Perbandingannya dengan SPNRG Lainnya

Chandra Gondowasito – NIM : 13504100

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if114100@students.if.itb.ac.id

Abstraksi

Mersenne Twister(MT) merupakan sebuah *pseudorandom number generator*(PNRG) yang dikembangkan oleh Makoto Matsumoto and Takuji Nishimura pada tahun 1996/1997. Kelebihan PNRG ini dibandingkan dengan PNRG lainnya adalah MT memiliki *period* yang sangat besar(2^{19937}-1), menghasilkan bilangan acak yang memiliki distribusi yang sangat bagus, pembangkitan bilangannya sangat cepat dan menggunakan memori yang efisien. Meskipun MT sebenarnya bukan termasuk dalam Secure PNRG, MT bisa dimodifikasi agar aman dipakai dalam bidang kriptografi. Makalah ini membahas pemodifikasian MT agar menjadi SPNRG, pemanfaatannya dalam bidang kriptografi dan perbandingan kekuatannya dengan beberapa algoritma SPNRG yang lainnya yang umum digunakan dalam kripografi.

Kata kunci: Mersenne Twister, Secure Pseudo Random Number Generator, PNRG, SPNRG, CSPNRG, Kriptanalisis, inisialisasi, seed

Pendahuluan

Random Number Generator merupakan bagian yang penting dalam aplikasi kriptografi. Meskipun telah terdapat pembangkit bilangan acak pada tiap compiler, pembangkit ini dapat dipastikan tidak cukup aman untuk digunakan dalam bidang kriptografi dan bahkan juga tidak menghasilkan bilangan yang benar-benar dapat dikatakan acak . Pembangkit bilangan acak ini memang tidak sepenuhnya acak karena memang tidak ditujukan untuk keacakan, Kebanyakan aplikasi sederhana seperti game pada computer hanya memerlukan sedikit bilangan acak sehingga tidak terlalu kentara ketidakacakannya. Di lain pihak, kriptografi sangat sensitive sekali terhadap sifat keacakan yang dihasilkan dari pembangkit acak ini. Dengan menggunakan pembangkit bilangan acak yang yang kurang bagus dapat diperoleh hasil dan hubungan yang aneh dimana hubungan semacam ini merupakan hal terburuk dari kriptografi.

Masalahnya adalah pembangkit bilangan acak tidak menghasilkan urutan acak, bahkan mungkin sama sekali tidak acak.Tentu saja mustahil menghasilkan sesuatu yang benar-benar acak dari computer.Komputer termasuk hal yang dapat diprediksi dari hubungan antara masukan, proses, dan keluaran. Dengan memasukkan dua masukan yang sama terhadap operasi yang sama akan dihasilkan keluaran yang sama pula.Komputer hanya berada pada kondisi yang terbatas dalam (meskipun jumlah kondisi tersebut besar namun tetap saja terbatas). Keluaran dari computer selau merupakan fungsi terhingga yang dapat dianalisis dari masulkan dan state dari computer. Hal ini berarti bahwa sembarang pembangkit bilangan acak pada computer memiliki periode. Segala hal yang memiliki periode tentu saja dapat diprediksi. Dan jika sesuatu bisa diprediksi, pastilah bukan acak. Sebuah pembangkit bilangan acak memerlukan masukan yang acak yang tidak dapat dihasilkan oleh computer itu sendiri.

Bilangan Acak

Hal terbaik yang dapat dihasilkan dari computer adalah pembangkit urutan bilangan acak semu. Urutan acak semu ini nampak seperti acak dalam hal ini periodenya haruslah cukup besar sehingga urutan bilangan yang dihasilkannya dalam waktu yang terbatas (yang digunakan) tidak memiliki periode. Suatu urutan bilangan daikatakan acak semu bila memiliki property berikut:

1. Nampak acak.

Hal ini berarti urutannya mampu melewati tes statistic keacakan yang dapat didefiniskan. Banyak usaha telah dilakukan untuk menghasilkan urutan bilangan acak semu pada computer, baik dari segi akademis ataupun tes keacakan. Semuanya tetap priodik, namun dengan periode minimal 2^{512} talah cukup bisa dipakau dalam berbagai aplikasi. Masalahnya masih pada hasil dan hubungan yang aneh. Tiap pembangkit urutan bilangan acak semu pasti akan menghasilkannya dalam kondisi tertentu yang dalam hal ini memberikan celah bagi kriptanalis untu kmenyerang system.

Criptographically Secure PseudoRandom Sequenmce

Aplikasi kriptografi bergantung sangat besar pada pembangkit urutan bilangan acak semu daripada aplikasi lainnya. Keacakan secara kriptografi bukan hanay berarti keacakan secara statistic, hal ini hanya salah satu kriterianya. Agar urutan bilangan dapat dikatakan sebagai bilangan acak semu yang aman dari sisi kriptografi, property ini harus dipenuhi:

2. Keterurutannya tidak dapat diprediksi.

Haruslah tidak mungkin secara komputasi untuk memprediksi bit acak yang akan dihasilkan dimana telah diketahui algoritma atau perangkat kerasnya dan semua bit yang dihasilkan sebelumnya.

Urutan bilangan acak semu yang aman secara kriptografi harusnya tidak dapat dikompres kecuali diketahui key(seed) nya yang dipakai untuk mengeset initial state dari pembangkit.

Serupa dengan algoritma kriptografi lainnya, pembangkit acak semu kriptografi juga menjadi subyek untuk diserang. Membuat pembangkit yang tahan terhadap serangan adalah inti dari kriptografi.

Masalah-masaah dari pembangkit acak pada komputer

Pada kenyataannya, keluaran dari pambangkit bilangan acak semu memiliki artifak yang membuatnya gagal melewati uji statistic pendeteksi pola. Diantaranya masalah artifak yang dimiliki adalah :

- Periode yang lebih pendek akibat kondisi seed tertentu(kurang dari periode yang sebenarnya dimiliki), seed ini biasa disebut sebagai week seed
- Urutan keluarannya memiliki distribusi dimensional yang jelek
- Adanya hubungan untuk nilai yang akan dihasilkan selanjutnya
- Beberapa urutan bit yang dihasilkan lebih acak daripada yang lainnya
- Kurangnya keseragaman dalam distribusi keluaran yangn dihasilkannya

Kelemahan yang biasa dimiliki oleh pembangkit acak semu yang jelek berkisar mulai hal yang sangat jelas sampai hal yang sangat abstrak. Misalnya, algoritma pembangkit bilangan acak RANDU yang telah dipakai berpuluh-puluh tahun pada computer mainframe ternyata sangat jelek, dan penilitian dimasa itu sangatlah kurang dari yang seharusnya dilakukan dalam menghadapi masalah ini. Biasanya, masalah dalam model desain dapat diketahui secara dini sehingga desainnya dapat diperbaiki sebelum benr-benar digunakan.

Usaha Awal Dalam Mendesain Pembangkit Bilangan Bulat Acak

Pembangkit bilangan acak semu dari computer pertama diberikan oleh John von Neumann pada tahun 1946 yang dikenal sebagai *middle-square method*. Algoritmanya sangat sederhana : pilih sembarang bilangan, kuadratkan, buang digit ditengah dari hasil pengkuadratan dan bilangan inilah sebagai bilangan acak, selanjutnya gunakan bilangan lainnya sebagai bilangana wal untuk algoritma ini lagi..

Masalah dengan *middle square method* adalah semua urutan bilangan yang dihasilkannya seketika itu pula berulang sendiri, beberapa bahkan sangat cepat, seperti bilangan 0000. Meskipun hal ini telah diketahui, namun pendekatan ini telah memuaskan untuk tujuan pada waktu itu dan khawatir apabila pendekatan matematis untuk memperbaikinya akan hanya menutupi kekurangan tersebut. *Middle-square method* telah tidak lagi digunakan dengan munculnya berbagai metode baru yang lebih baik

Cryptographically Secure Pseudorandom Number Generators

Suatu Pembangkit bilangan acak semu yang cocok digunakan dalam bidang kriptografi disebut sebagai *cryptographically secure PRNG* (CSPRNG). Perbedaan utama antara PRNG biasa dengan CSPRNG hanyalah satu: CSPRNG haruslah nampak acak dari berbagai jenis analisis dann pemeriksaan sedangkan PRNG hanya memerlukan keacakan dari sisi uji statistik. Terdapat skema desain pembangkit acak PNRG namun bukan termasuk kuat dari sisi kriptografi.

Beberapa kelas ari CSPRNGs adalah:

- Cipher aliran atau or cipher blok yang berjalan pada counter atau output feedback mode.

- Desain khusus yang telah kekuatannya secara matematis. MIsalnya adalah Blum Blum Shub, dimana
- PRNG yang telah didesain khusus sehingga aman secara kriptografi seperi fungsi CryptGenRandompada Cryptographic Application Programming Interface dari Microsoft dan algoritma Yarrow yang digunakan pada Mac OS X dan FreeBSD.
- Kombinasi dari beberapa PRNG yang bertujuan menghilangkan sifat ketidakacakan yang mungkin dimiliki suatu PNRG. Termasuk dalam kategori ini adalah Fortuna.

Mersenne Twister

Pada tahun 1997 , penemuan algoritma Mersenne twister oleh Matsumoto and Takuji Nishimura, menghilangkan masalah ayng dihadapai oleh pembangkit-pembangkit acak semu sebelumnya. Mersenne Twister memiliki periode yang sangat besar yaitu 2^{19937}-1 iterasi yang tidak akan selesai dihitung secara komputasi selamanya.. Keunggulan lainnya adalah memiliki keseragaman keluaran hingga dimensi 623 (untuk yang 32 bit), dan waktu eksekusinya lebih cepat daripada pembangkit acak yang baik lainnya. Mersenne Twister sekaran ini telah menjadi pilihan utama sebagai pembangkit bilangan acak semu untuk simulasi dan pemodelan

Meskipun Mersenne Twister sesuai untuk dipakai dalam banya kaplikasi, MT yang murni tidak cocok dipakai dalam kriptografi. Masalahnya dalah MT serupa dengan generalized feedback shift register sehingga keluaran yang dihasilkan dapat diprediksi dengan mudah.

Desain Mersenne Twister

Meersenne Twister menghasilkan suatu urutan bilangan semu acak yang selanjutnya disebut sebagai word dengan menggunakan rekursi lanjar Gaussian Field GF(2):

$$w_{624+i} = w_{397+i} \oplus ((w_i \& \texttt{0x80000000}) | (w_{1+i} \& \texttt{0x7fffffff}))A \quad (i = 0, 1, 2, \ldots).$$

Dengan w_i (i=0,1,2,…) adalah bilangan bulat 32 bit, yang masing masing dianggap sebagai sebuiah vector baris berdimensi 32 terhadap dua *field* elemen GF(2). Operator biner ? ? melambangkan operasi bit XOR yaitu penambahan sebagai vector.

Notasi heksadesimal 0x80000000 merupakan vector yang komponennya adalah nol semuanya kecuali bit satu yang terkiri. Jadim $((w_i \& 0x8000000) | (w_{1+i} \& 0x7fffffff))$ adalah vector baris yang diperoleh dengan menyambungkan MSB dari w_i dan dengan semua bit dari w_{1+I} kecuali MSB-nya. Vektor ini kemudian dikalikan dengan MAtriks A berukuran 32x32. MAtriks A ini adlaah berbentuk

$$\begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{31} & a_{30} & \cdots & \cdots & a_0 \end{pmatrix},$$

Sehingga perkalian ini dihitung dengan

xA = shiftkanan(x) (jika LSB dari x adalah 0)

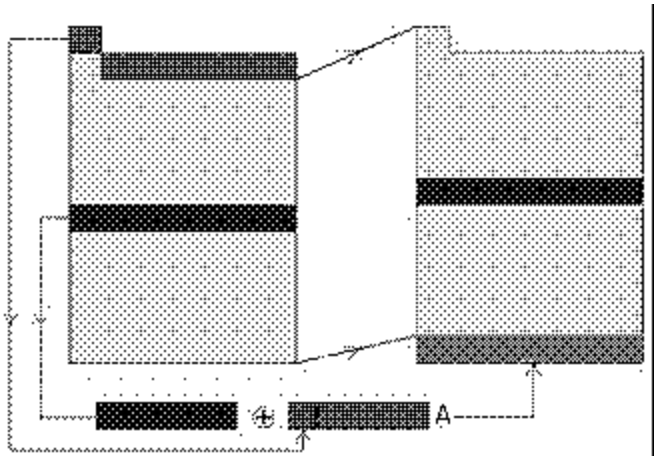
atau

xA = shiftkanan(x)? a (jika LSB dari x adalah 1)

Konstanta ini dipilih sehingga periode dari bilangan yang adihasilkan adalah 2^{19937}-1. Jumlah dari elemen yang bukan nol pada karakterisitik polinom dari fungsi transisi adalah 153.

Gambar berikut menggambarkan state transisi dari MT. State nya terdiri atas 623 word + 1 bit. State selanjutnya diperoleh dengan menggeser satu word ke atas, dan menyelipkan sebuah word baru yang dihitung secara lanjar dari bagian yang dibuanga dan elemen bagian tengah. Pada

implementasi pada perangkat lunak, digunakan teknik round-robin (yaitu memanfaatkan pointer) untuk masalah efifiensi pembangkitan bilangan.



Kelebihan dari konfigurasi ini dibandingkan dengan LFSR biasa dengan okefisien misalnya pada GF(2^{32}) adalah:

1. Tidak diperlukan operasi yang mahal seperti polynomial perkalian modulo
2. terdapat algoritma yang secara efisien mengecek periode maksimal dengan teori Galois,
3. Kecepatan dalam menghasilkan bilangan tidak bergantung pada a, yang membuat pencarian parameter menjadi lebuiih mudah. Hal ini berbeda dengan LFSR dimana kecepatan menghasilkan bilangannya bergantung pada pemilihan koefien tertentu.

Untuk menginisialisasi Mersenne Twister, diperlukan bilangan $w_0,w_1,w_2,\ldots,w_{623}$ sebagai state awal , dimana semua 31 bit dari w_0 kecuali MSB-nya diacuhkan dalam menghasilkan word selanjutnya. Jadi, ruang state memiliki 624x32-31 = 19937 bit. Urutan keluaran dari Mersenne Twister adalah $w_{624},w_{625},\ldots$, yaitu MT melewatkan iswi dari state awal.

Inisialisasi pada MT adlaah sangat penting karena sifat kelanjaran dan kejarangan pada rekursinya. Jika state awal memiliki terlalu banyak bit nol, maka urutan bilangan keluaran akan memiliki kecenderungan yang sama untuk keluaran yang melebihi 1000. Pertama, $w_0,w_1,\ldots,w_{623}$ diset nilainya dengan nilai yang rumit melalui rekursi

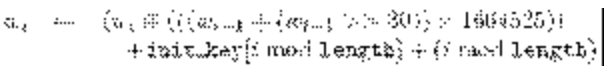
$$\begin{aligned} w_0 &\leftarrow 19650218, \\ w_i &\leftarrow ((w_{i-1} \oplus (w_{i-1} >> 30)) + i) \times 1812433253 \end{aligned}$$

(1= i = 623), dengan w_i dianggap sebagai variable bilangan bulat integer 32 bit unsigned, dan setiap operasi aritmatika adalah dalam modulo 2_{32}. Notasi >> 30 berarti pergeseran bit ke kanan sebanyak 30 bit (Konstanta 19650218 adalah hari ulang tahun salah satu pencipta Mersenne Twister. Konstanta 1812433252 adalah pengali untuk pembangkit kongruen lanjar, dipilih acak tanpa alasan tertentu).

Rekursi ini dipilih sehingga memiliki property difusi bit yang bagus. Perkalian dengan konstanta memiliki peroperti pencampuran bit yang bagus kecuali bnahwa difusi untuk informasi bit dilakukan dari kanan ke kiri. Dua bit paling sifnifikan (yang mengumpulkan informasi dari semua bit setelah perkalian) dikirimkan ke kedua LSB bit dari w_i dengan meng-XOR-kan untuk mengomplemenkan perkalian. Lambang operator assignment yang dipakai menandakan bijeksi.

Penambahan dengan I bermanfaat untuk menghindari fenomena berikut. Misalkan I tidak ditambahkan pada rekursi pada inisialisasi, misalkan pula bahwa nilai awal w_0 dipilih sembarang pula dan $w_0,\ldots,w_{623}$ adalah urutan keluaran. Apabila pada inisialisasi yang lain dipakai nilai awal w_0', yang akan menghasilkan keluaran $w_0',\ldots,w_{623}'$. Yang menjadi masalah adalah mungkin terjadi bahwaq $w_0'=w_1$ secara kebetulan (atau $w_0'=w_2$ atau semacamnya) maka $w_i'=w_i+1$ untuk i=0,1,…,622. Kesamaan tersebut pada nilai awal menghasilkan keluaran yang saling berhuungan untuk 100 keluaran atau lebih. Penambahan dengan i menghindari fenomena ini.

Nilai awal diberikan sebagai array init_key[length] dengan length sembarang hingga 64. Skema inisialisasi init_by_array menggantikan $w_1,\ldots,w_{623}$ dengan substitusi rekusi berikut:



Untuk i=1,2,…,623. perhatikan bahwa tiap perkalian atau penjumlahan dilakukan dalam modulo 1<<32. Rekursi ini dipilih dengan alas an sama seperti sebelumnya. Alasan mengambil I modulo length pada rekursi terakhir adalah karena misalkan tidak digunakan odulo length, misalkan pula satu inisialisasi diberikan oleh sebuah array init_key[] dan inisialisasi lain dengan array lainnyadengan panjang dua kalinya dengan isinya adalah perulangan array awal. Maka kedua inisialisasi akan menghasilkan keadaan yang sama. Hal ini dihindari dengan melakukan modulo length.

Selanjutnya , word pertama disubstitusikan

$w_0 \leftarrow w_{623,}$

kemudian dilakukan kembali substitusi rekursi yang sama

Untuk i=1,2,…,623.. Selanjutnya MSB dari w_0 diset satu, untuk menghindari keadaan awal nol.

Inisialisasi ini memilikiproperti sebaran yang bik. Perubahan satu bit pada array inisialisasi init_key[] merubah inisial state secara drastic. Kunci lemah yang mungkin hanyalahpada kunci-kunci ayng memiliki perbedaan hanya pada word terakhir pada init_key[]. Namun., karena keluaran dari Mersenne Twister dimulai dari w_{624}, yang bergantung pada w_{397}, yang nampaknya sulit dikendalikan karena sekurang-kurangnya dilakukan 397-(64+64) kali perkalian pada inisialisasi pada word terakhir dari init_key[]. (Angka 64+64 ini adlaah karena ukuran kunci dan ukuran dari nilai awal adalah maksimal 64). Sebagai tambahan, tiap word dari keadaan internal ditransformnasi dengan rekursi nonlanjar pada kunci. Napaknya sangat sulit menggunakan teknik kriptanalisis diferensial terhadap kunci..

Inisialisasi tidak didesain untuk kegunaan kriptografi meskipun nampaknya memiliki cukup ketahanan.

Berbagai kelas serangan terhadap PNRG

1. .Serangan Kriptanalisis Langsung

Ketika penyerang secara langsung mampu membedakan antara keluaran PNRG dan keluaran acak, hal ini disebut kriptanalisis langsung. Serangan semacam ini dapat diaplikasikan pada kebanyakan tetapi tidaksemua penggunaan PNRG. Misalnya, PNRG yang hanya digunakan untuk menghasilkan kunci-kunci Triple-DES mungkin tidak akan pernah rentan terhadap serangan ini karena keluaran dari PNRG tidak pernah terlihat secara langsung

2. Serangan Berbasiskan Masukan

Serangan Masukan muncul ketika penyerang mampu menggunakan pengetahuan atau kendali akan masukan bagi PNRG untuk mngkriptanalis PNRG tersebut, yaitu membedakan antara keluaran dari PNRG dan nilai acak biasa.

Seranagn berbasis masukan ini dapat lebih lanjut dibagi menjadi serangan known-input, replayed-input, dan chosen-input. Serangan chosen-input mungkin praktis digunakan terhadap smardcards dan lain-lainnya yang kebal terhadap serangan kriptanalis secara fisik; juga bias praktis dipakai untuk aplikasi tang menangani masukan pesan, password yang dipilih pengguna, statistic jaringan, dan lain-lain, terhadap PNRG-nya sebagai

sample entropi. Serangan replayed-input juga nampak praktis digunakan pada situasi yang sama, tapi memerlukan kendali yang lebih kurang atau kerumitan pada bagian penyerang. Serangan known-input praktis dipakai dalam sembarang situasi dimana beberapa masukan untuk PNRG diprediksikan oleh pendesain system akan sulit ditebak, ternyata mudah ditebak pada kasus tertentu

3. Serangan Pengenalan Adanya State

Serangan PEngenalan Adanya State berupaya memperluas keberhasilan usaha sebelumnya dalam menyerang S sejauh mungkin. Misalkan bahwa, untuk alas an apapun, penetrasi temporer pada system computer , kesuksesan kriptanalisis, dan lain-lain, penyerang berhasil mempelajari state internal S pada point tertentu di waktu. Serangan ini berhasil ketika penyerang mampu mendapatkan kembali keluaran PNRG yang tidak diketahui (atau membedakan keluaran dari PNRG adri data acak) dari sebelum S diperoleh atau mendapatkan kembali keluaran setelah PNRG telah mengumpulkan sejumlah urutan masukan yang penyerang tidak bias menebak.

Serangan Pengenalan State akan berjalan dengan baik bila PNRG dimulai pada state yang tidak aman(dapat ditebak) karena kurangnya entropi untuk memulai. Serangan ini dapat pulka bekerja ketika S telah dikenali oleh sembarang serangan lainnya atau dengan sembarang cara lainnya. Dalam pralkteknya, adalah baik menganggap bahwa state S telah dikenal oleh penyerang; untuk menjaga keberlangsungan system, PNRG haruslah kebal terhadap kelanjutan serangan state ini selengkapnya.

a. Serangan Backtracking

   Serangan backtracking menggunakan pengenalan state S dari PNRG pada waktu t untuk mempelajari keluaran PNRG sebelumnya.

b. Serangan Pengenalan Permanen

   Serangan pengenalan permanent terjadi bila sekali penyerang mengetahui state S pada waktu t, semua nilai S pada masa depan, dan masa lampau rawan diserang.

c. Serangan Tebakan Iteratif.

   Serangan Tebakan iteratif menggunakan pengetahuan akan S pada waktu t, dan memperngaruhi keluaran PNRG, untuk mempelajari S pada waktu t+E , ketika masukan yang dikumpulkan selama masa ini dapat ditebak(tapi tidak diketahui) oleh penyerang

d. Serangan Bertemu di Tengah

Serangan bertemu di tengah adala hkombinasi dari serangan tebakan iterative dengan serangan backtracking. Pengetahuan akan S pada waktu t dan t+2E akan membuat penyerang mengetahui S pada waktu t+E

Panduan Dalam Menggunakan PNRG yang tidak aman secara Kriptografi

Meskipun Mersenne Twister tidak aman digunakan dalam kriptografi karena berbasiskan pada rekursi lanjar. Setiap sekuen bilangan acak semua yang dihasilkan dari pembangkit melalui rekursi lanjar adalah tidak amana karena dari sejumlah hasil keluaran yang cukup banyak dapat diprediksi kelanjutan keluarannya.

Manun, mengingat berbagai kelebihan yang dimilikinya, sangat sayang bila MT tidak dimanfaatkan. Terdapat beberapa panduan dalam menggunakan PNRG yang tidak aman secara kriptografi seperti MT ini agar tahan terhadap serangan kriptanalisis sehingga sebenarnya bias saja PNRG diubah sehingga mendekati sebuah *Secure* PNRG. Beberapa panduan untuk memanfaatkan PNRG dalam aplikasi kriptografi adalah :

1. Menggunakan fungs hash untuk melindungi keluaran PNRG dari serangan.

Jika PNRG dianggap rawan serangan kriptanalis, maka keluaran dari PNRG seharusnya diproses degan fungs ihash kriptografi. Tidak semua kekurangan dari PNRG akan teramankan setelah dilakukan hash, jadi melakukan hash tidak menjamin keamanan. Meskipun demikian, hash akan meningkatkan keamanan PNRG jauh lebih baik.

2. Melakukan hash pada masukan PNRG dengan sebuah counter atau timestamp sebelum dipakai.

Untuk mencegah serangan chosen-attack yang umum, masukan seharusnya du lakukan hash dengan sebuah timestamp atau counter sebelum dumimasukkan ke dalam proses pembangkitan PNRG. Bila hal ini terlalu mahal untuk dilakukan pada setiap masukan yang akan diproses maka pendesain system bias hanya melakukan hash terhadap masukan yang dianggap dapat dikendalikan oleh penyerang.

3. Dalam periode tertentu membangkitkan nilai state awal yang baru bagi PNRG.

Untuk PNRG seperti ANSI X9.17 yang membiarkan sebagian besar state nya tidak berubah sekali telah diniisialisasi, state PNRG yang benar-benar baru seharusnya secara berkala dibangkitkan lagi dari PNRG yang sekarang. Hal ini memastikan bahwa sembarang PNRG dapat secara mandiri mengeset nilai awalnya diberikan waktu yang cukup lama dan entropi amsukan

4. Memperhatikan dengan seksama permulaan PNRG dan nilai awal.

Cara terbaik untuk melindungi dari semua serangan berbasiskan pengamatan state adalah dengan tidak membiarkan state PNRG diketahui. Sementara adalah tidak mungkin untuk memastikan hal ini,pendesain system haruslah menghabiskan banyak usaha untuk emmulai PNRG nya dari titik yang tidak dapat ditebak, menangani file-file yang berisi nilai awal(seed) dari PNRG secara cerdik, dan lain-lain.

Panduan Dalam Mendesain PNRG

Dengan berbekal pengetahuan akan berbagai jenis serangan yang mungkin dilakukan, adalah cukup beralasan bahwa kita bias mencoba mendesain sebuah PNRG baru yang bias kebal terhadap serangan-serangan. Berikut ini adaah beberapa panduan yang kiranya dapat digunakan untuk membangun PNRG yang kebal serangan:

1. Landaskan PNRG tersebut pada sesuatu yang terbukti kuat.

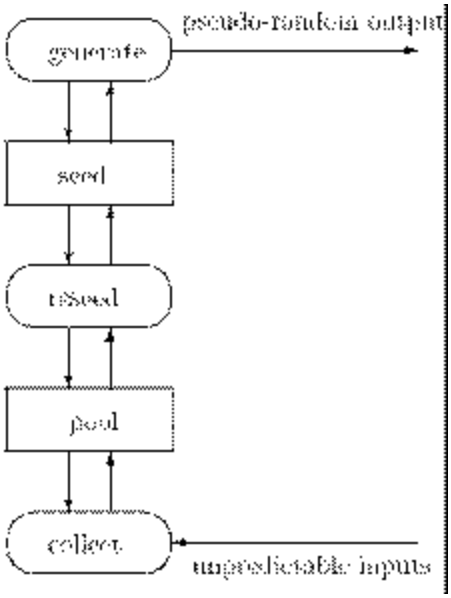
PNRG haruslah didesain sehingga serangan yang berhasil terhadapnya secara langsung berakibat keberhasilan serangan pada beberap primitive kriptografi yang dipercayai adalah kuat. Secara nalar, hal ini haruslah terbukti.

2. Pastikan bahwa seluruh state dari PNRG berbubah sepanjang waktu.

Keseluruhan state internall yang rahasia haruslah berubah sepanjang waktu. Hal ini mencegah diketahuinya state tunggal yang tidak pernah berubah.

3. Lakukan inisialisasi ulang secara brutal pada PNRG.

Bagian dari keadaan internal ayng digunakan untuk menghasilkan keluaran haruslah dipisahkan dari *entropi pool*. State pembangkit haruslah diubah hanya ketika cukup entropi telah dikumpulkan untuk mencegah serangan penebakan berdasarkan iterasi, berdasarkan perkiraan konservatif. Gambar berikut mengilustrasikan arsitektur yang mungkin untuk mengimplementasikan penginisialisasian ulang secara brutal ini.



4. Tahan terhadap backtracking.

PNRG haruslah didesain untuk tahan terhadap backtracking. Idealnya, hal ini berarti bahwa keluaran t tidak dapat diperkirakan secara praktis oleh penyerang yang mengetahui state dari PNRG pada waktu t+1. Dapat pula dilakukan dengan melewatkan state dari PNRG pada fungsi hash satu arah pada tiap jeda keluaran, sehingga membatasi ruang gerak serangan backtracking.

5. Tahan terhadap serangan Chosen-Input.

Masukan untuk PNRG haruslah dikombinasikan dengan state dari PNRG sedemikian rupa sehingga bila diberikan urutan masukan yang tidak dapat diprediksi, penyerang yang di awal mengetahui state PNRG namu ntidak mengetahui urutan masukan, dan penyerang yang awalnya mengetahui urutan masukan namun tidak tahu state PNRG, keduanya tidak dapat menebak state akhir. Hal ini menyediakan perlindungan terhadap baik serangan chosen-input ataupun pengenalan state.

6. Sembuh dari pengenalan state secara cepat.

PNRG dapat mengambil manfaat dari tiap bit entropi dari masukan yang diterimanya. Penyerang yang menginginkan memperlajari efek dari state PNRG terhadap urutan masukan haruslah menebak seluruh urutan masukan.

Pemodifikasian Mersenne Twister Sehingga Mungkin Menjadi SPNRG

Beradsarkan panduan desain di atas, sekarang diterapkan panduan tersebuit terhadap Mersenne Twister. Sistem cipher aliran digunakan untuk mengahsilkan PNRG yang aman secara kriptografi ari kunci dan melakukan XOR terhadap "plain message" untuk memperoleh "ciphermessage". Salah astu cara untuk menghasilkan CSPNRG semaacm itu adalah dengan menggunkan sebuah pembangkit yang tidak aman seperti LFSR (yaitu Mersenne Twister), diinisialisasi dengan kunci kemudian memfilter keluaran, yaitu dengan melakukan beberapa fungsi yang rumit untuk memperoleh urutan bilangan yang aman. Salah satu cara untuk menghasilkan PNRG yang aman secara kriptografi adalah menggunakan MT dan mengompres keluaran darinya dengan menggunakan fungsi hash seperti SHA1 atau MD5.

MT menghasilkan keluaran bilangan bulat berukuran 32 bit yang dianggap sebagai unsigned(yang disebut sebagai word). Kunci yang diberikan ditambah dikonkatenasi dengan nilai awal dan dilewatkan pada skema inisialisasi yang telah dijelaskan sebelumnya dari Mersenne Twister. Pertama disiapkan dahulu variable accum berukuran word, yang diset nilainya dengan 1 pada awal(tidak harus satu, yang penting ganjil).

Selanjutnya dilakukan iterasi berikut agar keluaran 8 bit yang dihasilkan adalah CSPN:

1. Hasilkan satu bilangan acak gen_rand dengan menggunakan Mersenne Twister.
2. Kalikan bilangan tersebut dengan accum:

   accum←accumx(gen_rand|1)

3. Keluarkan 8 bit MASB dari accum . Kembali ke langkah 1.

Untuk meningkatkan keamanan, 64 byte pertama dari keluaran tidak digunakan.

Disini, notasi bahasa C : "|" menandakan operasi OR. Operasi ini bertujuan membuat pengali ganjil(bila tidak maka setelah beberapa iterasi, accum akan menjadi nol). Pewrkalian adalah dalam modulo 1<<32.

Cara ini menghasilkan sebuah byte bilangan acak-semu, yang sesuyai dengan persyaratan untuk cipher aliran. Cipher aliran ini selanjutnya dinamakan sebagai CriptMT yang merupakan kepanjangan dari Criptografic Mersenne Twister. Ukuran dari internal state dari MT nampaknya membuat tidak bisanya dilakukan serangan memori-waktui.

Jika semua bit dari accum digunakan (berbeda dari CripMT) maka keluaran yang dihasilkan bukanlah aman secara kriptografi karena pengubahan pada accum akan membuat diketahuinya keluaran dari MT (kecuali untuk bit LSB nya), sehingga edngan menggunakan aljabar lanjar dapat ditentukan keadaan internal setelah mengamati 19937 bit dari keluaran. Namun, jika hanya 8 bit MSB setelah dilakukan perkalian yang diamati, tidak dapat dibayangkan bagaimana cara mengetahui state internal dari MT.

Adalah penting untuk menggunakan hanya MSB: LSB bit selalu 1 dan bit kebua dari accum bertemu dengan hasil penjumlahan (modulo) dari bit kedua dari ekluaran adri MT sajauh ini, dimana dapat dihitung keadaan internal dari MT. MSB nampaknya aman digunakan, karenapola difusi bit dari perkalian adalah dari kanan ke kiri dan MSB mengumpulkan informasi darisemua LSB bit dari kedua operan: accum dan keluaran dari MT.

Keamanan dari CripMT bergantung pada MT dan inisialisasinya. Dengan fakta bahwa:

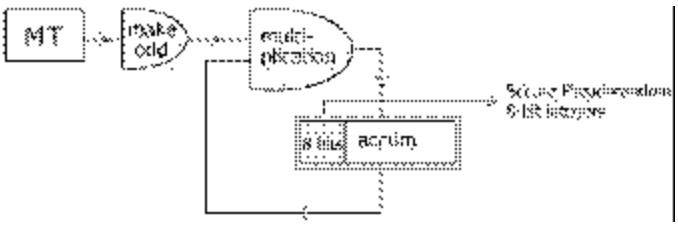
1. Ukuran state internal dari MT sangat besar.
2. ¾ dari keluaran MT tidak dipakai
3. MSB setelah dilakukan perkalian berisi informasi dari semua bit, dan
4. Inisialisasi sangat nirlanjar

Harusnya berdampak pada keamanan yang tinggi, namun tentu saja hal ini perlu dipelajari dan dianalisis lebih lanjut. Periode dari tiap 8 bit keluaran dari CriptMT adalah tetap 2^{19937} -1.

Nalar dari desain CriptMT ini adalah:

1. MEnggunakan pembangkit lanjar yang cepat, yang memiliki state yang sangat besar(yaitu ribuan bit)
2. Memfilter keluaran dengan otomata Finite-State nirlanjar yang memiliki state yang kecil(satu word)
3. Hanya bagian kecil dari informasi dari state yang dikeluarkan (yaitu 8 bit dari 32 bit) seperti yang digambarkan berikut

Ngubahan. Point satu memastikan periode yang besar, dan Point 3 memastikan difusi bit yang rumit dengan tambahan kecepatan yang cukup baik.



Perbandingan dengan CSPNRG Lainnya

Blum-Blum-Shub pseudorandom number generator

Blum Blum Shub (BBS) adalah sebuah pseudorandom number generator yang dikembangkan pada tahun 1986 oleh Lenore Blum, Manuel Blum and Michael Shub.

BBS memiliki bentuk algoritma::

$x_{n+1} = (x_n)^2 \bmod M$

dengan $M = pq$ adalah perkalian dari dua bilangan prima yang sangat besar p and q. Pada tiap langkah dari algortima, beberapa keluaran diambil dari x_n; keluaran ini biasanya adalah parity bit dari x_n atau satu atau lebih LSB dari x_n.

Kedua bilangan prima, p and q, haruslah kongruen 3 (mod 4) (untuk memastikan bahwa tiap residu kuadratik memiliki satu akar kuadrat yang juga merupakan residu kuadratik) dan and gcd(Ï†(p-1), Ï†(q-1)) haruslah kecil (untuk membuat lebar siklus besar).

Karakterisitk yang menarik dari BBS generator adalah kemungkinan untuk menghitung sembarang nilai x_i secara langsung:

$$x_i = \left(x_0^{2^i \bmod (p-1)(q-1)}\right) \bmod M.$$

Keamanan BBS

Pembangkit ini tidak sesuai digunakan dalam simulasi, hanya baik digunakan untuk bidang kriptografi, karena tidak cukup cepat dalam menghasilkan bilangan acak semu. Namun, BBS memiliki bukti kekuatan yaitu yang berhubungan dengan kualitas pembangkit dari sisi kesulitan komputasi untuk melakukan faktorisasi bilangan bulat yang besar. Bila bilangan prima dipilih dengan benar dan cukup besar, dan O(log log M) bit LSB dari tiap x_n dikeluarkan, maka dengan basis batas pada pertumbuhan M yang semakin besar, membedakan bit-bit keluaran dari kumpulan acak akan sekurang-kurangnya sesulit memfaktorkan M.

Jika faktorisasi bilangan bulat masih sulit dilakukan secara efisisen(seperti yang diperkirakan), maka BBS dengan M yang sangat besar akan memiliki keluaran yang bebas dari pola tak acak yang bisa disingkap dalam waktu perhitungan yang memungkinkan. Hal ini membuatnya sekokoh teknologi enkripsi lainnya yang berlandaskan pada masalah pemfaktoran bilangan besar, seperti RSA.

CryptGenRandom

CryptGenRandom merupakan sebuah fungsi pembangkit bilangan acak yang terdapat pada Cryptographic Application Programming Interface dari Microsoft. Microsoft merekomendasikan penggunaannya pada semua perangkat lunak dimana keamanan adalah isu yang penting.

Metode Operasi

Penyedia layanan kriptografi dari Microsoft memiliki implementasi yang sama dari CryptGenRandom, yang sekarang didasarkan pada fungsi internal yang disebut RtlGenRandom. Hanya sekelumit bagian adri algoritmanya yang dipublikasikan pada tahun 2006:

RtlGenRandom menghasilkan bilangan acak yang sesuai dengan spesifikasi FIPS 186-2 appendix 3.1 dengan memanfaatkan SHA-1 sebagai fungsi G – nya, dan dengan entropi diperoleh dari:

- ID proses saat ini (GetCurrentProcessID).
- ID thread saat ini (GetCurrentThreadID).
- Tick count sejak waktu boot (GetTickCount).
- Waktu saat ini (GetLocalTime).

- Berbagai counter dengan performansi tinggi (QueryPerformanceCounter).
- MD4 hash berdasarkan blok lingkungan penggunayang didalamnya termasuk username, nama komputer, dan *search path*.
- Counter CPU dengan presisi tinggi seperti RDTSC, RDMSR, RDPMC

Algoritma yang spesifik belum dipubliaksikan sehingga tidak mungkin bagi peneliti untuk melaukan review dan evaluasi keefektifannya. Kelemahan teoritis yang dapat dilihat adalah penggunaan algoritma yang telah usang (seperti MD4), dan ketergantungan pada perolehan entropi dari counter yang secara monoton bertambah nilainya yang bisa diperkirakan atau dikendalikan oleh penyerang yang memiliki akses local pada mesin.

Yarrow

Algoritma Yarroew merupakan Secure PNRG yang didesain oleh Bruce Schneier, John Kelsey, dan Niels Ferguson dari laboratorium Counterpane Labs.

Yarrow adalah PNRG yang efisien, dan sangat aman untuk Windows, Windows NT, dan UNIX. Yarrow menyediakan bilangan acak untuk berbagai jenis aplikasi kriptografi seperti ennkripsi, signature, integrity,dan lain-lain.

Yarrow dikembangkan untuk memnuhi kebutuhan adanya Secure PNRG yang berdasarkan penelitian. Desain Yarrow yang telah diperbaikai ,Fortuna membuat Yarrow kurang dipakai lagi.

Fortuna

Fortuna adalah CSPNRGyang diusulkan oleh Bruce Schneier dan Niels Ferguson. Namanya diambil dari nama Fortuna, dewi keberuntungan Romawi.

Fortuna merupakan keluarga dari Secure PRNGs; desainnya memberikan kesempatan pada implementor. Fortuna terbagi atas beberap abgian:

- Pembangkit, yang sekali diinisialisasi akan menghasilkan keluaran data acak semua yang tak terbatas.
- Akumulator entropi, yang mengumpulkan data acak asli dari berbagai sumber dan menggunakannya untuk menginisialisasi ulang pembangkit ketika dirasa telah terkumpul cukup data acak.
- File seed yang menyimpan cukup state yang memungkinkan computer untuk memulai menghasilkan bilangan acak seketika setelah di boot.

Pembangkit ini didasarkan pada semua cipher blok yang bagus, disarankan menggunakan AES, Serpent atau Twofish. Ide dasarnya adalah menjalankan cipher dalam mode counter, mengenkripsi nilai yang berurutan dari counter yang bertambah terus nilainya. Dengan hanya ini bias menghasilkan penyimpangan statistic dari keacakan;misalnya, menghasilkan 2^{65} blok acak asli akan menghasilkan rata-rata sepasang blok yang identik, namun tidak ada blok yang berulang sama sekali dari 2^{128} data pertama yang dihasilkan oleh cipher 128 bitpada mode counter. Sehingga, kunci diubah secara berkala : tidak lebih dari 1MB data dihasilkan tanpa penggantian kunci. Kunci juga diganti setiap permintaan data sehingga pengenalan kunci tidak akan membahayakan keluaran bilangan acak sebelumnya.

Akumulator entropi didesain untuk tahan terhadap serangan injeksi, tanpa memerlukan perkiraan entropi yang rumit dan tidak terandalkan. Terdapat beberapa sumber untuk entropi, masing-masing mendistribusikan entropinya ke penampungan entropi dan pada inisialisasi ulang ke n pada pembangkit, penampung k digunakan hanya jika 2^k habis membagi n. Jadi, penampung ke k

hanya digunakan $1/2^k$.. Semakin besar jumlah penampung, maka:

1. Berpartisipasi pada inisialisasi ulang lebih jarang, namun
2. Mengumpulkan sejumlah besar entropi antara jeda waktu inisisalisasi ulang. Inisialisasi dilakukan dengan melakukan hashing pada nilai entropi penampung tertentu menjadi kunci blok cipher menggunakan dua iterasi dari SHA-256.

Keamanan Fortuna

Kecuali penyerang mampu mengendalikan semua sumber data entropi yang mengalir ke system dimana tidak ada algoritma yang dapat menyimpan nilai ini,maka terdapat suatu nilai k dimana penampung k mengumpulkan entropi yang cukup antara waktu reseeding sehingga penginisialisaian ulang dengan menggunakan penampung tersebut meningkatkan keamanan. .Dan, penampung tersebut digunakan pada jeda waktu proporsional dengan jumlah entropi yang diperlukan. Maka, system akan selalu sembuh dari serangan injeksi dan waktu yang diperlukan untuk sembuh maksimal adalah konstanta yang lebih besar dari waktu teoritis yang bias diambilnya jika bias mengidentiikasi sumber penampung mana yang rusak dan mana yang tidak. Penentuan ini bergantung pada ada tidaknya cukup penampung, Fortuna menggunakan 32 penampung dan mengharuskan penginisialisasian ulang maksimal sepuluh kali tiap detik. Kehabisan penampung akan memerlukan waktu 13 tahun, waktu yang cukup untuk kelangsungan hidup data.. Implementor yang paranoid atau yang memerlukan penghasilan data acak pada waktu yang cepat dengan inisialisasi yang sering bias menggunakan jumlah penampung yang lebih besar.

Fortuna berbeda dengan algoritma Yarrow terutama pada penanganan akumulator entropi. Yarrow memerlukan bahwa tiap entropi didampingi mekanisme untuk menentukan jumlah eksak entropi yang diberikan, dan hanya menggunakan dua penampungdan menggunakan hash SHA1 daripada iterasi menggunakan SHA-256.

Simpulan

Mersenne Twister bukan termasuk kelas CSPNRG karena berbasiskan pada rekursi lanjar. Namun mengingat banyaknya kelebihan yang dimilikinya adalah bermanfaat menggubahnya menjadi CSPNRG. Untuk mengubahnya terutama dimabfaatkan fungsi LSFR dan fungsi hash SHA1.

CSPNRG dari Mersenne Twister ini memiliki property berikut yang mungkin membuatnya benar-benar CSPNRG:

1. Ukuran state internal dari MT sangat besar.
2. ¾ dari keluaran MT tidak dipakai
3. MSB setelah dilakukan perkalian berisi informasi dari semua bit, dan
4. Inisialisasi sangat nirlanjar

Bila dibandingkan dengan berbagai CSPNRG ayng telah umum digunakan, Mersenne Twister hasil modifikasi ini paling unggul bila digunakan dalam aplikasi yang tidak hanya memerlukan keamanan namun juga disrtibusi bilangan acak yang bagus dan kecepatan dalam menghasilkan bilangan acak. CriptMT juga nampaknya kebal terhadap serangan pengenalan state karena penggunaan memori yang kecil

Blum-blum shub memiliki kelebihan seperi RSA yaitu memanfaatkan property susahnya memfaktorkan bilangan besae, sehingga keamanannya sama dengan aplikasi yang mengandalakan algoritma seperti RSA. Degan menggunakan bilangan prima yang lebih besar, kekuatannya meningkat. Dibandingkan dengan MT, BBS lebih aman namun kekurangannya dalah tidak dapat digunakan selain di bidang aplikasi

kriptografi, tidak bias dipakai pada aplikas seperti permodelan dan aplikasi yang memerlukan efisiensi.

CryptGenRandom merupakan fungsi kripgografi yang disediakan Microsoft untuk digunakan dalam perangkat lunak. Keamanannya dipertanyakan karena algoritmanya tidak dipublikasikan dan karena penggunaan beberapa algoritma klasik yang keamanannya kurang seperti MD4. Akan lebih baik bila algoritmanya dipublikasikan sehingga dapat direview dan dianalisis tentang kekuatannya. Dibandingkan dengan MT, CryptRandom nampaknya lebih mangkus.

Yarrow merupakan algoritma pertama hasil penelitian. Fortuna menggnatikan Yarrow dengan metode penampungan yang baik. Keamanannya bergantung pada banyaknya sumber data yang digunakan untuk menginisialisasi seed. MT nampaknya tidak bias dibandingkan dengan algoritma ini karena keamanannya belum diuji.

Pustaka

- Knuth, D. E. The Art of Computer Programming. Vol. 2. Seminumerical Algorithms 3rd Ed. Addison-Wesley, Reading, Mass., (1997).
- Matsumoto, M. and Nishimura, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation, 8 (1998) 3–30.
- Matsumoto, M. and Nishimura, Cryptographic Mersenne Twister and Fubuki Stream/Block Cipher
- Matsumoto, M. and Nishimura, T. Mersenne Twister Homepage. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
- http://www.forth.org.ru/~mlg/mirror/home.earthlink.net/~neilbawd/mersenne.html
- http://www.quadibloc.com/crypto/co4814.htm
- http://eiffelzone.com/esd/mersenne/index.html
- National Institute for Standards and Technology, \Secure Hash Standard,"
- NIST FIPS PUB 180, U.S. Department of Commerce, 1993.
- [NIST94] National Institute for Standards and Technology, \Digital Signature Standard,"
- NIST FIPS PUB 186, U.S. Department of Commerce, 1994.
- [OW95] P.C. van Oorschot and M.J. Wiener, \Parallel collision search with application to hash function and discrete logarithms," *2nd ACM Conf. on Computer and Communications Security*, New York, NY, ACM, 1994.
- [OW96] P.C. van Oorschot and M.J. Wiener, \Improving implementable meetin the-middle attacks by orders of magnitude," *CRYPTO '96*, Springer-Verlag, 1996.
- [Plu94] C. Plumb, \Truly Random Numbers, *Dr. Dobbs Journal*, v. 19, n. 13, Nov 1994, pp. 113-115.
- [Ric92] M. Richterm \Ein Raus chgenerator zur Gweinnung won quasi-idealen Zufallszahlen fur die stochastische Simulation," Ph.D. dissertation, Aachen University of Technology, 1992. (In German.)
- [RSA94] RSA Laboratories, RSAREF cryptographic library, Mar 1994,ftp://ftp.funet.fi/pub/crypt/cryptography/asymmetric/rsa/rsaref2.tar.gz
- [SV86] M. Santha and U.V. Vazirani, \Generating Quasi-Random Sequences from Slightly Random Sources," *Journal of Computer and System Sciences*, v. 33, 1986, pp. 75{87.
- [Sch96] B. Schneier, *Applied Cryptrography*, John Wiley & Sons, 1996.
- [Zim95] P. Zimmermann, *The O_cial PGP User's Guide*, MIT Press, 1995.
- GG.B. Agnew, \Random Source for Cryptographic Systems," *Advances in Cryptology | EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 77{81.

- [ANSI85] ANSI X 9.17 (Revised), \American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985.
- [Bal96] R.W. Baldwin, \Proper Initialization for the BSAFE Random Number Generator," *RSA Laboratories Bulletin*, n. 3, 25 Jan 1996.
- [Dai97] W. Dai, Crypto++ library, http://www.eskimo.com/~weidai/cryptlib.html.
- [DIF94] D. Davis, R. Ihaka, and P. Fenstermacher,\Cryptographic Randomness from Air Turbulience in Disk Drives," *Advances in Cryptology CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 114{120.
- [ECS94] D. Eastlake, S.D. Crocker, and J.I. Schiller, \Randomness Requirements for Security," RFC 1750, Internet Engineering Task Force, Dec. 1994.
- [FMK85] R.C. Fairchild, R.L. Mortenson, and K.B. Koulthart, \An LSI Random Number Generator (RNG)," *Advances in Cryptology: Proceedings of CRYPTO '84*, Springer-Verlag, 1985, pp. 203{230.
- [Gud85] M. Gude, \Concept for a High-Performance Random Number Generator Based on Physical Random Noise," *Frequenz*, v. 39, 1985, pp. 187{190.
- [Gut98] P. Gutmann, \Software Generation of Random Numbers for Cryptographic Purposes," Proceedings of the 1998 Usenix Security Symposium, 1998, to appear.
- [Koc95] P. Kocher, post to sci.crypt Internet newsgroup (message-ID pckDIr4Ar.L4z@netcom.com), 4 Dec 1995.
- [LMS93] J.B. Lacy, D.P. Mitchell, and W.M. Schell, \CryptoLib: Cryptography in Software," *USENIX Security Symposium IV Proceedings*, USENIX Association, 1993, pp. 237{246.
- [NIST92] National Institute for Standards and Technology, \Key Management Using X9.17," NIST FIPS PUB 171, U.S. Department of Commerce, 1992.