STUDI DAN PERBANDINGAN *CSPRNG BLUM BLUM SHUB* DAN *YARROW*

Fajar Yuliawan – NIM: 13503022

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13022@students.if.itb.ac.id

Abstrak

Bilangan acak banyak sekali digunakan dalam kritografi, misalnya untuk umpan (*seeds*) untuk fungsi-fungsi kriptografi yang membangkitkan nilai-nilai matematis seperti bilangan prima besar pada *RSA* dan *ElGamal*, *initialization vectors* untuk enkripsi dengan mode *cipher block chaining*, dan nilai acak untuk berbagai skema tanda tangan digital, seperti *DSA*. Sayangnya, bilangan acak yang benar-benar acak (*true random numbers*) sangat sulit dibangkitkan, terutama dengan menggunakan komputer yang dirancang secara *deterministic*. Oleh karena itu, dilakukan pendekatan dengan membangkitkan bilangan acak semu (*pseudo-random numbers*). Algoritma yang digunakan untuk membangkitkan bilangan acak semu dinamakan *pseudo-random number generators* (*PRNG*).

Untuk tujuan kriptografi, *PRNG* yang digunakan harus benar-benar aman, artinya jika algoritma *PRNG* dan sederetan bilangan-bilangan acak yang telah dibangkitkan sudah diketahui, bilangan-bilangan yang dibangkitkan sebelumnya dan sesudahnya harus tidak dapat diprediksi dengan algoritma komputasi dengan waktu polinomial (*unpredictable to the left and right*). *PRNG* semacam ini dinamakan *CSPRNG* atau *cryptographically secure pseudo-random number generators*. Salah satu *CSPRNG* yang paling sederhana dan paling mangkus adalah *CSPRNG Blum Blum Shub* (*BBS*). Keamanan *CSPRNG* ini terletak pada landasan teori bilangan yang kuat, yaitu sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima dan juga sulitnya memecahkan *qudratic residuosity problem*. Satu *CSPRNG* lagi yang menggunakan pendekatan yang jauh berbeda dibandingkan *BBS* adalah *Yarrow-160*. *Yarrow* membangkitkan bilangan acak dengan memperhitungkan *entropy* inputnya.

Pada makalah ini, akan dijelaskan mengenai *CSPRNG Blum Blum Shub* dan *Yarrow-160*, hal-hal yang mendasarinya dan perancangan algoritmanya. Selanjutnya akan dilakukan perbandingan kedua *CSPRNG* tersebut dari segi algoritma, kerumitan komputasi, performansi dan keamanan bilangan-bilangan acak yang dihasilkan.

Keywords: *Pseudo-Random Number Generators*, *PRNG, Cryptographically Secure Pseudo-Random Number Generators, CSPRNG, Blum Blum Shub, Yarrow, Yarrow-160.*

1 Pendahuluan

Manusia sudah menggunakan pembangkit bilangan acak selama ribuan tahun, misalnya dalam permainan yang berkaitan dengan peluang. Contoh yang paling sederhana adalah dadu dengan 6 sisi. Banyak orang sudah memiliki pemahaman intuitif bahwa setiap sisi dadu akan muncul sekitar 1/6 kali. Pemahaman yang lain adalah jika dadu dilempar sebanyak 6 kali, ada kemungkinan ada satu angka yang tidak muncul sama sekali. Dari pemahaman intuitif tersebut, pembangkit bilangan acak dapat didefinisikan sebagai alat yang menghasilkan sederetan bilangan-bilangan yang tidak dapat diprediksi sebelum bilangan tersebut dibangkitkan. Dengan definisi tersebut, dadu dengan 6 sisi dapat dianggap sebagai sebuah pembangkit bilangan acak. Dengan menggunakan dadu, seorang pengamat tidak

dapat memprediksi sebuah angka yang muncul dengan peluang yang lebih baik daripada 1/6.

Lebih umum, jika bilangan yang dibangkitkan terletak diantara 1 ... *n*, maka seorang pengamat tidak dapat memprediksi bilangan yang muncul dengan peluang lebih dari 1/*n*. Dan jika *m* bilangan sudah dibangkitkan, kemudian seorang pengamat diberitahu nilai-nilai *m* – 1 bilangan yang dibangkitkan sebelumnya, ia tetap tidak dapat memprediksi bilangan yang dibangkitkan ke-*m* dengan peluang lebih baik daripada 1/*n*.

Bruce Schenier menggunakan tiga hal sebagai definisi dari pembangkit bilangan acak:

1. *Output* yang dihasilkan kelihatan acak (*looks random*), artinya lolos uji keacakan statistik.
2. Tidak mudah diprediksi (*unpredictable*), artinya walaupun algoritma dan bilangan-bilangan acak sebelumnya sudah diketahui, bilangan acak yang dihasilkan berikutnya harus tidak dapat dikomputasi dengan mudah.
3. Tidak dapat dihasilkan ulang (*cannot be reliably reproduced*), artinya jika dijalankan dua kali dengan *input* yang tepat sama, maka *output* yang dihasilkan benar-benar berbeda.

Pembangkit bilangan acak yang memenuhi tiga syarat di atas dinamakan *true random number generators* (*TRNG*). Sayangnya, bilangan acak yang benar-benar acak (*true random numbers*) sangat sulit dibangkitkan, terutama dengan menggunakan komputer yang dirancang secara *deterministic*. Syarat ketiga di atas hampir tidak mungkin dipenuhi oleh komputer. Oleh karena itu, yang bisa dilakukan adalah membangkitkan bilangan acak semu (*pseudo-random number*). Hal ini dapat dilakukan karena pembangkit bilangan acak semu termasuk *deterministic finite state machine*.

Yang dimaksud dengan acak dalam bilangan acak semu adalah bilangannya tidak mudah diprediksi. Bilangan acak semu pada umumnya dihasilkan dengan rumus-rumus matematika dan biasanya bilangan acak yang dibangkitkan dapat berulang kembali secara periodik. Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (*PRNG*).

Salah satu contoh *PRNG* adalah pembangkit bilangan acak kongruen lanjar (*linear congruential generator* atau *LCG*). *LCG* adalah salah satu pembangkit bilangan acak tertua dan sangat terkenal. *LCG* didefinisikan dalam relasi rekurens:

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

yang dalam hal ini,

x_n = bilangan acak ke-*n* dari deretnya
x_{n-1} = bilangan acak sebelumnya
a = faktor pengali
b = *increment*
m = modulus
(*a*, *b*, dan *m* semuanya konstanta)

Kunci pembangkit adalah x_0 yang disebut **umpan** (*seed*). Dalam hal ini x_0 bersifat rahasia.

Secara teoritis, *LCG* mampu menghasilkan bilangan acak yang lumayan baik, namun ia sangat sensitif terhadap pemilihan nilai-nilai *a*, *b*, dan *m*. Pemilihan nilai-nilai yang buruk dapat mengarah pada implementasi *LCG* yang tidak bagus, yaitu bilangan acak yang dihasilkan dapat diprediksi urutan kemunculannya. Oleh karena itu, *LCG* tidak aman digunakan untuk kriptografi. Namun demikian, *LCG* tetap berguna untuk aplikasi non-kriptografi seperti simulasi, karena kecepatannya dalam membangkitkan bilangan acak dan hanya membutuhkan sedikit operasi bit. Selain itu *LCG* mangkus dan memperlihatkan sifat statistik yang bagus dan sangat tepat untuk uji-uji empirik.

Dalam kriptografi, bilangan acak banyak dibutuhkan antara lain untuk:

1. *Session* dan *message keys* dalam *cipher* simetri seperti *triple-DES* atau *Blowfish*.
2. Umpan (*seeds*) untuk fungsi-fungsi yang membangkitkan nilai-nilai matematis seperti bilangan prima besar pada *RSA* dan *ElGamal*.
3. Dikombinasikan dengan password untuk mengacaukan program untuk menebak password secara *offline*.
4. *Initialization Vectors* untuk enkripsi dengan mode *cipher block chaining*.
5. Nilai acak untuk berbagai skema tanda tangan digital, seperti *DSA*.
6. *Random challenges* pada protokol otentikasi seperti Kerberos

Untuk itu, pembangkit bilangan acak yang digunakan harus dapat menghasilkan bilangan

yang tidak mudah diprediksi oleh pihak lawan. Pembangkit bilangan acak semacam ini dinamakan *cryptographically secure pseudo-random number generator* (*CSPRNG*). Persyaratan *CSPRNG* adalah:

1. Secara statistik ia mempunyai sifat-sifat yang bagus, yaitu lolos uji keacakan statistik.
2. Tahan terhadap serangan yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan.

Untuk persyaratan yang kedua ini, maka *CSPRNG* hendaklah memenuhi dua persyaratan sebagai berikut:

1. Setiap *CSPRNG* seharusnya memenuhi "uji bit-berikutnya" (*next-bit test*) sebagai berikut: Diberikan k buah bit barisan acak, maka tidak ada algoritma dalam waktu polinomial yang dapat memprediksi bit ke-(k+1) dengan peluang keberhasilan lebih dari 1/2.
2. Setiap *CSPRNG* dapat menahan "perluasan status", yaitu jika sebagian atau semua statusnya dapat diungkap (atau diterka dengan benar) maka tidak mungkin merekonstruksi aliran bilangan acak.

Kebanyakan *PRNG* tidak cocok digunakan untuk *CSPRNG* karena tidak memenuhi kedua persyaratan *CSPRNG* yang disebutkan di atas. *CSPRNG* dirancang untuk tahan terhadap bermacam-macam kriptanalis.

Perancangan *CSPRNG* dapat dibagi menjadi beberapa kelompok.

1. Perancangan berbasis primitif kriptografi

   Algoritma *cipher* blok yang aman dapat dikonversi menjadi *CSPRNG* dengan mode cacah. Ini dilakukan dengan memilih kunci acak dan mengenkripsi 0, mengenkripsi 1, mengenkripsi 2, dan seterusnya (pencacahan juga dapat dimulai dari bilangan selain 0). Jelaslah bahwa periodenya akan menjadi 2^n untuk blok berukuran n-bit. Nilai-nilai awal tidak boleh diketahui oleh pihak lawan. Kebanyakan algoritma *cipher* aliran membangkitkan aliran bit kunci yang dikombinasikan dengan plainteks (umumnya di-*XOR*-kan); aliran kunci ini dapat juga digunakan sebagai *CSPRNG* yang bagus (meskipun tidak selalu demkian, seperti pada *RC4*).

   Salah satu *CSPRNG* yang berbasis *cipher* blok (menggunakan algoritma *DES*) dan diadopsi sebagai standar *FIPS* bekerja sebagai berikut:

   Masukan: D = informasi jam/tanggal, s = umpan yang berukuran 64-bit, K = kunci.

   Proses: Hitung $I = DES_K(D)$.

   Luaran: bilangan acak sekarang

   $$x = DES_K\ (I \text{ XOR } s)$$

   lalu mutakhirkan umpan s yang baru untuk bilangan acak berikutnya sebagai

   $$s = DES_K\ (x \text{ XOR } I).$$

2. Perancangan berbasis teori bilangan

   CSPRNG dapat juga dirancang berdasarkan persoalan matematika yang sulit, seperti pemfaktoran biilangan menjadi faktor prima, logaritma diskrit, dan sebagainya. Contoh dua *CSPRNG* yang berdasarkan teori bilangan adalah *Blum Blum Shub* dan modifikasi *RSA*.

   Blum Blum Shub (*BBS*) adalah *CSPRNG* yang paling sederhana dan paling mangkus (secara kompleksitas teoritis). *BBS* dibuat pada tahun 1986 oleh Lenore Blum, Manuel Blum dan Michael Shub. Algoritma *BBS* dapat dijelaskan secara singkat sebagai berikut:

   1. Pilih dua buah bilangan prima rahasia p dan q, yang masing-masing kongruen 3 modulo 4 (dalam praktek bilangan prima yang digunakan cukup besar).
   2. Kalikan keduanya menjadi $n = pq$. Bilangan n ini disebut **bilangan bulat Blum.**
   3. Pilih bilangan ulat acak lain, s, sebagai umpan sedemikian sehingga:
      (i) $2 \le s \le n$
      (ii) s dan n relatif prima
      kemudian hitung $x_0 = s^2 \bmod n$
   4. Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:

(i) hitung $x_i = x_{i-1}^2 \bmod n$
(ii) z_i = bit *LSB* dari x_i

5. Barisan bit acak yang dihasilkan adalah $z_1, z_2, z_3, ...$

Sedangkan *CSPRNG* berbasis *RSA* membangkitkan bilangan acak dengan algoritma sebagai berikut:

1. Pilih dua buah bilangan prima rahasia, p dan q dan bilangan bulat e yang relatif prima dengan $(p-1)(q-1)$.
2. Kalikan keduanya menjadi $n = pq$.
3. Pilih bilangan acak lain, s, sebagai x_0 yang dalam hal ini $2 \le s \le n$
4. Barisan bit acak dihasilkan dengan melakukan iterasi berikut sepanjang yang diinginkan:
   (i) hitung $x_i = x_{i-1}^e \bmod n$
   (ii) z_i = bit *LSB* dari x_i
5. Barisan bit acak yang dihasilkan adalah $z_1, z_2, z_3, ...$

Sama seperti halnya *BBS*, keamanan pembangkit acak modifikasi *RSA* ini terletak pada sulitnya memfaktorkan n. Jika n besar, maka pembangkit bilangan acak ini dikatakan aman.

3. Perancangan khusus

Salah satu *CSPRNG* dengan perancangan khusus adalah *Yarrow-160*. *Yarrow* dirancang menjadi empat bagian utama, yaitu

1. *Entropy Accumulator*
2. *Reseed Mechanism*
3. *Generation Mechanism*
4. *Reseed Control*

Pada makalah ini, akan dilakukan studi dan perbandingan *CSPRNG* berbasis teori bilangan yaitu *Blum Blum Shub* dan *CSPRNG* dengan perancangan khusus, yaitu *Yarrow-160*.

2 *Blum Blum Shub*

Di awal sudah dijelaskan secara singkat mengenai algoritma *CSPRNG Blum Blum Shub*. Selanjutnya akan dilakukan studi lebih jauh mengenai *CSPRNG* ini. Karena *Blum Blum Shub* merupakan *CSPRNG* yang dirancang dengan dasar teori bilangan, maka akan dijelaskan dasar teori bilangan *CSPRNG Blum Blum Shub*.

2.1 Beberapa Dasar Teori Bilangan *Blum Blum Shub*

Pertama, Teorema Sisa Cina (*Chinese Remainter Theorem* atau *CRT*) yang menyebutkan bahwa jika $n_1, n_2, ..., n_k$ bilangan-bilangan asli yang sepasang-sepasang saling relatif prima (artinya n_i dan n_j relatif prima untuk sembarang i,j dimana $i \neq j$) dan $a_1, a_2, ..., a_k$, sembarang bilangan-bilangan asli, maka ada bilangan bulat x yang memenuhi

$$x \equiv a_1 \bmod n_1$$
$$x \equiv a_2 \bmod n_2$$
$$...$$
$$x \equiv a_k \bmod n_k$$

Bukti: Misalkan $N = n_1 n_2 ... n_k$. Selanjutnya, definisikan

$$m_i = \frac{N}{n_i} = n_1 \ldots n_{i-1} n_{i+1} \ldots n_k$$

untuk $i = 1,2,...,k$. Karena $n_1, n_2, ..., n_k$ bilangan-bilangan asli yang sepasang-sepasang saling relatif prima, maka n_i dan m_i juga relatif prima, sehingga ada bilangan bulat x_i yang memenuhi

$$m_i x_i \equiv 1 \bmod n_i$$

untuk $i = 1,2,...,k$. Sehingga

$$a_i m_i x_i \equiv a_i \bmod n_i$$

Sekarang perhatikan bilangan

$$X = a_1 m_1 x_1 + ... + a_k m_k x_k$$

Bilangan tersebut memenuhi sistem persamaan lanjar yang diberikan.

□

Selanjutnya tentang konsep sisa kuadratik (*quadratic residues*) dan simbol *Legendre* (*Legendre symbol*). Kita definisikan untuk sembarang bilangan bulat n, kita definisikan $Z_n = \{1,2,...,n-1\}$. Suatu bilangan $a \in Z_n$ disebut sisa kuadratik jika *modulo n* jika ada bilangan $b \in Z_n$ yang memenuhi persamaan kongruensi

$$a \equiv b^2 \bmod n.$$

Selanjutnya, himpunan semua sisa kuadratik *modulo n* dilambangkan dengan QR_n. Lebih jauh, kita definisikan juga

$$QNR_n = Z_n \setminus QR_n.$$

Dengan kata lain, QNR_n adalah himpunan semua anggota Z_n yang bukan anggota QR_n. Sebagai contoh, untuk $n = 23$, perhatikan bahwa

$$1 \equiv 1^2 \bmod 23,$$
$$4 \equiv 2^2 \bmod 23,$$
$$9 \equiv 3^2 \bmod 23.$$

Dengan demikian, $1,4,9 \in QR_{23}$. Jika pengamatan ini dilakukan lebih jauh, didapatkan bahwa

$QR_{23} = \{1,2,3,4,6,8,9,12,13,16,18\}$ dan
$QNR_{23} = \{5,7,10,11,14,15,17,19,20,21,22\}$

Selanjutnya, kia definisikan *Legendre symbol*. Misalkan p bilangan prima ganjil. Jika $a \in Z_p$ simbol *Legendre* didefinisikan sebagai

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & p \mid a \\ 1 & a \in QR_p \\ -1 & a \notin QR_p \end{cases}$$

Teorema berikut berguna untuk menghitung *Legendre symbol* dari suatu bilangan $a \in Z_p$.

Jika p bilangan prima ganjil, dan $a \in Z_p$, maka

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p$$

Bukti: Misalkan $a \in QR_p$, maka $a \equiv b^2 \bmod p$ untuk suatu bilangan $b \in Z_p$. Dengan *Fermat Little Theorem*,

$$1 \equiv b^{p-1} \equiv \left(b^2\right)^{\left(\frac{p-1}{2}\right)} \equiv a^{\frac{p-1}{2}} \bmod p$$

Sekarang jika $a \in QNR_p$, misalkan g *generator* dari Z_p (karena Z_p merupakan *group* siklik dengan *order* $p - 1$ maka Z_p memiliki generator). Dengan demikian, $a \equiv g^t \bmod p$ untuk suatu bilangan asli t ganjil. Misalkan $t = 2s + 1$, sehingga

$$a^{\frac{p-1}{2}} \equiv \left(g^t\right)^{\frac{p-1}{2}} \equiv \left(g^{2s}\right)^{\frac{p-1}{2}} \cdot g^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \bmod p$$

Sekarang perhatikan bahwa

$$\left(g^{\frac{p-1}{2}}\right)^2 \equiv 1 \bmod p$$

sehingga

$$g^{\frac{p-1}{2}} \equiv 1 \bmod p \text{ atau } g^{\frac{p-1}{2}} \equiv -1 \bmod p$$

Namun karena g adalah generator dari Z_p, maka *order* dari g adalah $p - 1$ sehingga

$$g^{\frac{p-1}{2}} \equiv -1 \bmod p$$

Hal ini melengkapkan pembuktian.

□

Dengan demikian, kita punya teorema berikut:
Jika p bilangan prima ganjil, maka

$$|QR_p| = |QNR_p| = (p - 1)/2$$

Bukti: Misalkan g adalah generator dari Z_p. Dengan menggunakan bukti pada teorema sebelumnya $g^t \in Z_p$ jika dan hanya jika $t \in \{0,1,2,...,p - 2\}$ genap dan ini ada sebanyak $(p - 1)/2$.

□

Sifat lain dari *Legendre symbol* adalah jika p bilangan prima ganjil dan $a, b \in Z_p$, maka

$$\left(\frac{a}{n}\right) \cdot \left(\frac{b}{n}\right) = \left(\frac{ab}{n}\right) \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$$

Bukti: Hal ini merupakan akibat dari identitas berikut

$$a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} \equiv (ab)^{\frac{p-1}{2}} \bmod p$$

□

Sekarang kita definisikan *Jacobi symbol* yang sama dengan *Legendre symbol* tetapi untuk *modulo* bilangan komposit.

Misalkan n bilangan bulat ganjil dengan faktorisasi prima $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$. Jika $a \in Z_n$, maka *Jacobi symbol* didefinisikan sebagai

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \cdots \left(\frac{a}{p_k}\right)^{e_k}$$

Dan sama seperti *Legendre symbol*, *Jacobi symbol* memiliki sifat

$$\left(\frac{a}{n}\right)\cdot\left(\frac{b}{n}\right)=\left(\frac{ab}{n}\right)$$

Sekarang kita hitung banyaknya akar kuadrat dari setiap sisa kuadratik. Sebelumnya, kita buktikan dulu tiga fakta berikut.

1. Jika p bilangan prima dan $a \in Z_p$, maka $-a \equiv a \bmod p$ jika dan hanya jika $p = 2$.
2. Jika p bilangan prima ganjil dan $a,b \in Z_p$, maka $a^2 \equiv b^2 \bmod p$ jika dan hanya jika $a = b$ atau $a \equiv -b \bmod p$.
3. Misalkan $n = p_1p_2 \ldots p_k$, dimana $p_1, p_2, \ldots, p_k$ adalah bilangan-bilangan prima berbeda dan $k \geq 2$. Sebuah bilangan $a \in Z_p$ adalah sisa kuadratik *modulo* n jika dan hanya jika setiap komponen dari *CRT transform*nya yang sesuai dengan *modulo* $p_1, p_2, \ldots, p_k$ merupakan sisa kuadratik dari Z_{p_i}.

Dengan tiga hal tersebut, kita dapat membuktikan teorema berikut:

Misalkan n bilangan bulat ganjil. Jika $a \in QR_n$, maka banyaknya akar kuadrat berbeda dari a adalah tepat sebanyak 2^k, dimana k adalah banyaknya faktor prima berbeda dari n.

Bukti: Dari tiga fakta sebelumnya, kita dapat menyimpulkan bahwa $(\pm a_1, \pm a_2, \ldots, \pm a_k)$ adalah akar-akar kuadrat dari sebuah sisa kuadratik di Z_n. Dengan demikian, sebuah sisa kuadrat memiliki paling sedikit 2^k akar kuadrat. Namun sebuah sisa kuadratik di Z_p hanya memiliki dua akar kuadrat saja. Dengan demikian, ada tepat 2^k akar kuadrat.

□

Kita lanjutkan dengan teorema berikut:

Misalkan $n = pq$, dimana p dan q bilangan prima ganjil. Ada tepat setengah anggota Z_n yang memiliki *Jacobi symbol* +1 dan tepat setengah yang memiliki *Jacobi symbol* –1. Notasikan himpunan ini berturut-turut dengan notasi $Z_n(+1)$ dan $Z_n(-1)$. Tidak ada anggota $Z_n(-1)$ yang merupakan sisa kuadratik dan ada tepat setengah anggota $Z_n(+1)$ yang merupakan sisa kuadratik dari Z_n.

Bukti: Dengan menggunakan teorema sebelumnya, kita tahu bahwa ada tepat setengah anggota Z_p dan Z_q yang merupakan sisa kuadratik. Dengan menggunakan definisi *Jacobi symbol*, ada empat kemungkinan, yaitu (+1)(+1), (+1) (–1), (–1)(+1), dan (–1) (–1) untuk membentuk perkalian untuk menghitung *Jacobi symbol* anggota Z_n. Tapi hanya kemungkinan pertama saja yang memberikan sisa kuadratik. Tiga kemungkinan yang lain tidak memberikan sisa kuadratik.

□

Sekarang kita masuk ke aplikasi teorema-teorema tersebut dalam *CSPRNG Blum Blum Shub*. Kita definisikan terlebih dahulu bilangan prima *Blum* dan membuktikan beberapa sifat-sifat penting yang berkaitan dengan perancangan *CSPRNG Blum Blum Shub*.

Bilangan prima p adalah bilangan prima *Blum* jika $p \equiv 3 \bmod 4$. Salah satu sifat penting dari bilangan prima *Blum* adalah sebagai berikut:

Jika p bilangan prima ganjil, maka

$$-1 \in QNR_p \Leftrightarrow p \text{ bilangan prima } Blum$$

Bukti: Perhatikan bahwa

$$-1 \in QNR_p \Leftrightarrow -1 = \left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}} \bmod p$$

Dan $(-1)^{\frac{p-1}{2}} \equiv -1 \bmod p$ jika dan hanya jika $(p-1)/2$ bilangan ganjil jika dan hanya jika $p \equiv 3 \bmod 4$. Dengan kata lain, p adalah bilangan prima *Blum*.

□

Misalkan $n = pq$, dimana p dan q adalah bilangan *Blum*. Jika $a \in QR_n$, maka a memiliki tepat empat akar dan tepat satu merupakan anggota QR_n.

Bukti: Dengan menggunakan teorema sebelumnya, anggota $R_p(a) \in QR_p$ memiliki dua akar kuadrat di Z_p. Jika $a = g^{2s}$, maka kedua akar tersebut adalah $b = g^s$ dan $-b = -g^s$. Sekarang perhatikan bahwa

$$(-b)^{\frac{p-1}{2}} = (-1)^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}}$$

sehingga

$$\left(\frac{b}{p}\right) \neq \left(\frac{-b}{p}\right)$$

karena p adalah bilangan prima *Blum*. Dengan demikian, ada tepat satu diantara $\{b, -b\}$ yang

merupakan anggota QR_p. Hal yang sama juga berlaku untuk *modulo q*, dan empat akar kuadrat *modulo n* adalah empat "*Chinese combinations*" dari akar-akar *modulo p* dan *q*. Dan karena

$$a \in QR_n \Leftrightarrow R_p(a) \in QR_p \wedge R_q(a) \in QR_q$$

maka kesimpulan mengikuti.

□

Berikut adalah teorema yang terakhir:

Misalkan $n = pq$ dimana p dan q adalah dua buah bilangan prima *Blum*. Maka fungsi

$$f : QR_n \rightarrow QR_n$$
$$x \rightarrow x^2 \bmod n$$

adalah sebuah permutasi

Bukti: Dari teorema sebelumnya, kita tahu bahwa setiap sisa kuadratik *modulo n* memiliki tepat satu akar kuadrat yang juga merupakan sisa kuadratik. Akibatnya, fungsi tersebut merupakan fungsi bijektif yaitu fungsi satu-satu dan pada. Karena fungsi tersebut didefinisikan dari dan ke himpunan yang sama, maka fungsi tersebut merupakan sebuah permutasi.

□

2.2 Perancangan Algoritma *Blum Blum Shub*

Algoritma *Blum Blum Shub* telah dijelaskan sebelumnya. Pemilihan bilangan prima p dan q agar kongruen 3 *modulo* 4 karena bilangan prima tersebut (bilangan prima *Blum*) memiliki beberapa sifat yang diperlukan agar bilangan acak yang dibangkitkan sulit diprediksi. Sifat-sifat tersebut telah dijelaskan pada teorema-teorema sebelumnya dan akan dijelaskan aspek keamanannya pada bagian berikutnya. Sedangkan bilangan prima yang kongruen 1 *modulo* 4 telah diuji dan tidak memenuhi syarat-syarat *CSPRNG*.

Sebagai contoh, misalkan kita memilih $p = 7$ dan $q = 19$, sehingga $n = pq = 7.19 = 133$. Selanjutnya pilih $s = 100$ dan kita hitung $x_0 = s^2 \bmod 133 = 100^2 \bmod 133 = 25$. Barisan bit acak yang dihasilkan adalah sebagai berikut:

$x_1 = 25^2 \bmod 133 = 93$, sehingga $z_1 = 1$

$x_2 = 93^2 \bmod 133 = 4$, sehingga $z_2 = 0$
$x_3 = 4^2 \bmod 133 = 16$, sehingga $z_3 = 0$
$x_4 = 16^2 \bmod 133 = 123$, sehingga $z_4 = 1$

demikian seterusnya. Perhatikan bahwa nilai x_i yang mungkin hanya terletak antara 1 sampai 133 saja, sehingga pada suatu saat barisan tersebut akan berulang. Akibatnya, barisan bit yang dihasilkan pun juga akan berulang.

Jadi untuk nilai n yang kecil, maka *CSPRNG Blum Blum Shub* dapat dikatakan tidak aman, karena jika penyerang sudah mengetahui pola periodenya, maka tidak akan sulit untuk menebak bit yang dibangkitkan berikutnya. Sebenarnya untuk n besar sekalipun, jika pemilihan s tidak bagus, maka bisa terjadi periodenya kecil. Jika ini terjadi, penyerang juga bisa mengetahui barisan bit-bit yang dibangkitkan berikutnya.

Satu hal yang menarik dari pembangkit ini adalah bahwa sebenarnya, kita tidak perlu melakukan iterasi untuk mendapatkan bilangan acak jika p dan q sudah diketahui, sebab x_i dapat dihitung secara langsung dengan persamaan:

$$x_i = x_0^{2^i \bmod (p-1)(q-1)} \bmod n$$

Persamaan tersebut dapat dibuktikan sebagai berikut.

$$x_i = x_{i-1}^2 \bmod n = (x_{i-2}^2)^2 \bmod n = x_{i-2}^4 \bmod n$$
$$= \ldots = x_0^{2^i} \bmod n$$

Selanjutnya dengan teorema Euler

$$x_0^{2^i} \bmod n = x_0^{2^i \bmod (p-1)(q-1)} \bmod n$$

Teorema Euler di atas berlaku karena sebenarnya, $(p-1)(q-1) = \Phi(pq)$ dimana $\Phi(m)$ menyatakan fungsi Euler yaitu banyaknya bilangan antara 1 sampai n yang relatif prima terhadap n.

Bilangan acak juga tidak harus 1 bit *LSB* tetapi bisa juga j buah bit, dimana j adalah bilangan bulat positif yang tidak lebih dari $\log_2(\log_2 n)$. Perhatikan contoh berikut:

Misalkan kita memilih $p = 11351$ dan $q = 11987$ sehingga $n = pq = 136064437$. Kita pilih $s =$

80331757 dan $j = 4$ (j tidak melebihi $\log_2 (\log_2 136064437) = 4.75594$). Kita hitung $x_0 = 80331757^2 \bmod 136064437 = 131273718$. Barisan bit acak yang kita hasilkan sebagai berikut:

$x_1 = x_0^2 \bmod n = 131273718^2 \bmod 136064437 = 47497112$, sehingga $z_1 = 47497112 \equiv 8 \bmod 2^4 = 1000_{\text{basis 2}}$.

$x_2 = x_1^2 \bmod n = 47497112^2 \bmod 136064437 = 69993144$, sehingga $z_2 = 69993144 \equiv 8 \bmod 2^4 = 1000_{\text{basis 2}}$.

$x_3 = x_2^2 \bmod n = 69993144^2 \bmod 136064437 = 13810821$ sehingga $z_3 = 13810821 \equiv 5 \bmod 2^4 = 0101_{\text{basis 2}}$.

.....

Barisan blok bit acak yang dihasilkan adalah

1000 1000 0101 .....

Atau dalam basis 10:

8 8 5

2.3 Keamanan *CSPRNG Blum Blum Shub*

Keamanan *CSPRNG Blum Blum Shub* untuk kriptografi terjamin karena asumsi pada sebuah persoalan teori bilangan yang disebut *quadratic residuosity problem*. Persoalan tersebut didefinisikan bersamaan dengan konsep penyelesainya (*solver*) sebagai berikut:

Quadratic residuosity problem dengan parameter n dan x adalah menentukan $x \in Z_n(+1)$ apakah x merupakan sisa kuadratik atau tidak. Sebuah penyelesai (*solver*) untuk persoalan residu kuadratik adalah sebuah algoritma *poly-time* $A(n,x)$ yang meng-*output*-kan 1 jika dan hanya jika x adalah sisa kuadratik dalam $Z_n(+1)$ dan 0 jika tidak.

Sedangkan keamanan *CSPRNG Blum Blum Shub* didasarkan pada asumsi berikut:

Misalkan t adalah bilangan asli dan n perkalian dua buah bilangan prima ganjil berbeda, $A(n,x)$ adalah penyelesai (*solver*) untuk *quadratic residuosity problem* dan $s = \lceil \log_2 n \rceil$ adalah panjang n dalam basis 2. Maka untuk s yang cukup besar dan untuk semua kecuali $1/s^t$, bagian dari bilangan n dengan panjang s, peluang $A(n,x)$ benar dalam menentukan apakah x adalah sisa kuadratik dalam Z_n atau tidak untuk n tetap dan x dipilih secara seragam dari semua anggota $Z_n(+1)$ adalah kurang dari $1 - 1/s^t$.

Dalam tulisannya, Blum, Blum dan Shub membuktikan bahwa, dengan mengasumsikan bahwa faktor-faktor dari n merupakan syarat perlu untuk menentukan apakah x anggota $Z_n(+1)$ merupakan sisa kuadratik atau tidak, faktor-faktor ini diperlukan untuk memiliki sedikit keuntungan dalam menebak paritas $x_{-1} = \sqrt{x_0}$, dengan diberikan parameter n dan x_0 dalam waktu polinomial. Kita dapat menyimpulkan di sini bahwa menebak paritas bilangan acak yang dibangkitkan oleh *CSPRNG* dari kiri atau dari kanan sebenarnya adalah permasalahan yang sama (*equivalent*). Sebuah *PRNG* yang membangkitkan bilangan acak yang kelihatan acak (*looks random*) dari satu arah saja, tentu bukan merupakan *CSPRNG*.

Dalam membuktikan bahwa *CSPRNG Blum Blum Shub* memang benar-benar pembangkit bilangan acak yang aman dengan *modulo* asumsi *quadratic residuosity*, Blum, Blum dan Shub pertama kali menunjukkan bahwa keuntungan dalam menebak paritas sebuah bilangan acak dari arah kiri deretan dapat dikonversi dalam keuntungan untuk menentukan *quadratic residuosity*. Selanjutnya mereka menggunakan hasil yang telah didapatkan oleh Goldwasser dan Micali untuk menunjukkan hubungan antara pembangkit bilangan acak dengan asumsi *quadratic residuosity*.

Teorema utama yang dibuktikan oleh Blum, Blum dan Shub adalah sebagai berikut:

CSPRNG Blum Blum Shub adalah pembangkit bilangan acak yang tidak dapat diprediksi (*unpredictable*) atau *cryptographic secure*, artinya untuk setiap *probabilistic poly-time predicting algorithm* $A(n,x)$, dan bilangan asli t, A memiliki paling besar $1/s^t$ keuntungan untuk n dalam memprediksi deretan dari arah kiri, dimana s merupakan panjang n dalam basis 2, untuk n yang cukup besar dan untuk semua kecuali $1/s^t$ bilangan-bilangan n dengan panjang s.

3 *Yarrow*

Seperti pada *CSPRNG Blum Blum Shub*, pertama kali akan dibahas mengenai dasar-dasar perancangan *CSPRNG Yarrow*.

3.1 Prinsip-Prinsip Perancangan *Yarrow*

Yarrow bertujuan untuk membuat *PRNG* yang mudah digabungkan dengan sistem-sistem yang telah dibuah oleh perancang sistem dan lebih baik dalam menahan serangan-serangan dari pihak lawan. Ada beberapa batasan yang dipenuhi dalam perancangan *CSPRNG Yarrow*.

1. Semuanya harus efisien. Tidak ada untungnya membangun sebuah *PRNG* yang tidak akan digunakan karena menghambat performansi sistem terlalu banyak.
2. *Yarrow* harus mudah digunakan, sehingga seorang pemrogram yang tidak memiliki dasar kriptografi sekalipun memiliki kesempatan untuk menggunakan *PRNG* secara aman.
3. Ketika memungkinkan, *Yarrow* menggunakan kembali (*re-use*) hal-hal yang sudah ada.

Yarrow diciptakan dengan menggunakan proses perancangan yang berorientasi pada serangan dari pihak lawan. Hal ini berarti, *Yarrow* dirancang dengan memikirkan serangan-serangan yang mungkin sejak awal. *Cipher* blok dirancang secara rutin dengan cara ini, dengan struktur yang dimaksudkan untuk mengoptimalkan kekuatan dalam melawan serangan-serangan yang sudah sering digunakan seperti *differential* dan *linear cryptanalysis*. Perancangan *Yarrow* sangat difokuskan pada serangan-serangan yang potensial. Namun hal ini tetap diseimbangkan dengan batasan-batasan perancangan lainnya yang penting seperti performansi, fleksibilitas, kesederhanaan, kemudahan penggunaan, portabilitas dan bahkan isu-isu kelegalan menyangkut eksportibilitas *CSPRNG Yarrow*.

Sebagian besar bagian pekerjaan dalam pengembangan *Yarrow* adalah pembangunan *framework* yang baik untuk estimasi *entropy* dan *reseeding* karena dua hal inilah yang menjadi dasar utama *CSPRNG Yarrow*. Dua hal tersebut sangat penting untuk keamanan *PRNG*. Mekanisme kriptografi dalam *Yarrow* sendiri hanya terdiri dari penggunaan fungsi *hash* dan *cipher* blok. Namun hal tersebut sudah sangat baik dalam menahan serangan-serangan yang pernah diketahui.

Ada beberapa terminologi penting dalam perancangan *Yarrow*. Pertama adalah kunci *PRNG*. Setiap saat di satu titik, sebuah *PRNG* mengandung sebuah status internal yang digunakan untuk membangkitkan luaran yang acak semu (*pseudo-random output*). Status-status tersebut bersifat rahasia dan mengendalikan sebagian besar bagian pemrosesan. Setara dengan konsep *cipher*, status-status ini akan disebut kunci *PRNG* (*the key of PRNG*).

Untuk memutakhirkan kunci, *PRNG* perlu mengumpulkan input-input yang benar-benar acak (*truly random*) atau paling tidak, tidak diketahui, tidak dapat diprediksi atau dikendalikan oleh penyerang. Contoh yang sering digunakan misalnya waktu yang diperlukan untuk menekan tombol, atau gerak tetikus secara detail. Pada umumnya, akan ada banyak sekali input pada setiap waktu dan nilai setiap input tersebut relatif kecil. Input-input tersebut dinamakan *samples*.

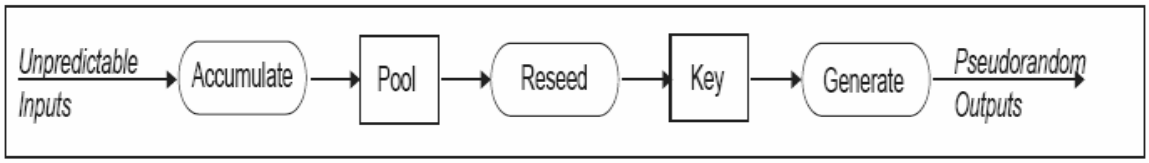
Pada banyak sistem, akan ada beberapa sumber yang menghasilkan *samples*. Oleh karena itu, *PRNG* akan mengelompokkan *samples* yang ada berdasarkan sumbernya.

Proses mengkombinasikan kunci *PRNG* yang ada dengan *samples* baru dinamakan *reseeding*. Proses tersebut akan menghasilkan kunci (status) baru.

Selanjutnya, bila sebuah sistem dimatikan atau di-*restart*, maka sistem akan menyimpan beberapa data dengan *entropy* tinggi misalkan kunci *PRNG* di dalam *non-volatile memory*, misalkan *hardisk*. Dengan demikian, ketika *PRNG* dihidupkan kembali, maka ia mulai berjalan dalam status yang tidak dapat ditebak. Data yang disimpan tersebut dinamakan *seed file*.

3.2 Komponen-komponen Perancangan *Yarrow*

Pada bagian ini, akan dibahas mengenai komponen-komponen *Yarrow*, dan bagaimana komponen-komponen tersebut berhubungan.



Gambar 1 Blok Diagram *Yarrow*

Salah satu prinsip perancangan *Yarrow* yang paling penting adalah sedapat mungkin, komponen-komponen tersebut saling bebas (*independent*). Dengan demikian, sistem-sistem dengan berbagai macam batasan perancangan tetap dapat menggunakan rancangan *Yarrow* secara umum.

Konsep utama dalam *Yarrow* adalah menggunakan komponen-komponen yang saling bebas secara algoritmik dalam perancangan level atas (*top level design*). Tujuannya bukan untuk menambah primitif-primitif keamanan untuk sistem kriptografik melainkan untuk memperluas primitif-primitif yang ada sebanyak mungkin. Oleh karena itu, dalam *Yarrow* hanya digunakan fungsi hash satu arah dan *cipher* blok. Dua hal tersebut merupakan primitif-primitif kriptografi yang paling sering dipelajari dan paling banyak digunakan.

Dalam *Yarrow* ada empat komponen utama, yaitu:

1. *Entropy Accumulator* yang bertugas mengumpulkan *samples* dari sumber-sumber *entropy* kemudian mengumpulkannya ke dalam dua *pools*.
2. *Generation Mechanism* yang membangkitkan *PRNG output* dari kunci-kunci yang ada.
3. *Reseed Machine* yang bertugas melakukan *reseed* kunci dengan *entropy* baru dari dalam *pools* secara periodik.
4. *Reseed Control* yang menentukan kapan *reseed* akan dilakukan.

Diagram blok *CSPRNG Yarrow* dapat dilihat pada gambar 1.

Selanjutnya akan dijelaskan peran masing-masing komponen pada rancangan *PRNG* yang lebih luas. Akan dijelaskan juga mengenai bagaimana cara tiap-tiap komponen tersebut berhubungan.

3.2.1 *Entropy Accumulator*

Entropy accumulation adalah sebuah proses dimana *PRNG* membutuhkan sebuah status internal yang baru dan tidak dapat ditebak (*unguessable*). Ketika inisialisasi *PRNG* dan *reseeding* dalam sebuah operasi, sangat penting untuk mengumpulkan *entropy* dari *samples* yang ada. Untuk menghindari serangan yang berupa tebakan secara iteratif dan juga tetap melakukan *reseeding PRNG* secara teratur, penting juga untuk mengestimasi *entropy* yang telah diperoleh sejauh ini. Mekanisme pengumpulan *entropy* juga harus tahan terhadap serangan yang berupa *input* yang dapat dipilih (*chosen-input attack*), artinya pihak lawan atau penyerang yang dapat mengendalikan beberapa *samples* (tetapi tidak semua *samples*) tidak dapat menyebabkan *PRNG* kehilangan *entropy* dari *samples* yang tidak diketahui.

Pada *Yarrow*, *entropy* dari *samples* dikukmpulkan ke dalam dua buah *pools*. Kedua *pools* tersebut disebut sebagai *fast pool* dan *slow pool*. *Fast pool* menyediakan *entropy* untuk *reseeding* secara teratur. Pada *slow pool*, *entropy* yang disediakan sangat jarang, tetapi berkelanjutan. Hal ini dimaksudkan untuk memastikan bahwa walaupun estimasi *entropy* sudah sangat *optimistic*, tetap ada sebuah *reseed* yang aman.

Tiap-tiap *pool* mengandung fungsi *hash* yang dijalankan pada setiap *input* yang masuk kedalamnya. Pada *Yarrow-160*, fungsi *hash* yang digunakan pada masing-masing *pool* adalah *SHA-1*, sehingga panjang *entropy* yang dapat dikumpulkan hanya 160-bit, tidak bisa lebih.

Satu isu yang penting dalam komponen *entropy accumulator* adalah estimasi *entropy*, yaitu proses untuk menentukan seberapa besar usaha yang harus dilakukan pihak lawan atau penyerang untuk menebak isi *pool* sekarang.

Metode *Yarrow* secara umum untuk melakukan estimasi *entropy* adalah dengan mengelompokkan *samples* berdasarkan sumber-sumbernya dan mengestimasi setiap *entropy* dari setiap sumber. Untuk melakukan hal ini, *Yarrow* menghitung *entropy* setiap *samples* secara terpisah, kemudian menjumlahkan perhitungan *entropy* yang berasal dari sumber yang sama.

Entropy dari tiap *samples* dihitung dengan menggunakan tiga cara:

1. Pemrogram menyediakan estimasi *entropy* suatu *sample* ketika dia menuliskan rutin program untuk mengumpulkan data dari sumber. Misalnya pemrogram mengirimkan sebuah *sample* dengan estimasi *entropy* 20 bits.
2. Untuk setiap sumber, sebuah estimator statistik khusus digunakan untuk mengestimasi *entropy* tiap *sample*. Uji ini dilakukan untuk mendeteksi situasi tak normal ketika *samples* memiliki *entropy* yang sangat rendah
3. Ada sebuah *system-wide maximum "density" samples* dengan memperhatikan panjang *sample* dalam bit, dan mengalikannya dengan suatu faktor konstanta yang kurang dari 1 untuk mendapatkan estimasi maksimum dari sebuah entropy dalam *samples*. Pada *Yarrow-160*, faktor pengali yang digunakan adalah 0,5.

Estimasi *entropy* dari *sample* yang dignakan adalah yang paling kecil diantara ketiga cara estimasi di atas.

Uji statistik yang spesifik yang digunakan tergantung pada kondisi sumber dan dapat berbeda-beda pada setiap implementasi.

3.2.2 *Generating Pseudorandom Outputs*

Generation mechanism menyediakan *output* dari *PRNG*. *Output* yang dihasilkan harus memiliki sifat bahwa, jika pihak lawan atau penyerang tidak mengetaui kunci *PRNG*, maka dia tidak dapat membedakan antara *output PRNG* dengan barisan bit-bit acak yang benar-benar acak (*truly random sequence of bits*).

Generation mechanism harus memenuhi syarat-syarat berikut:

1. Tahan terhadap berbagai serangan *cryptanalysis*
2. Efisien
3. Tahan terhadap *backtracking* setelah *key compromise*.
4. Mampu membangkitkan deretan *output* yang sangat panjang secara aman walapun tidak melalui proses *reseeding.*

3.2.3 *Reseed Mechanism*

Reseed mechanism bekerja untuk menghubungkan antara *entropy accumulator* dengan *generating mechanism*. Ketika *reseed control* menentukan bahwa *reseed* diperlukan, maka komponen *reseeding* harus memutakhirkan kunci yang digunakan oleh *generating mechanism* dengan informasi dari salah satu atau kedua *pools* yang dikelola oleh *entropy accumulator*. Dengan demikian, jika kunci atau *pool* tidak diketahui oleh pihak lawan atau penyerang sebelum *reseed* maka kunci baru yang dihasilkan setelah *reseed* juga tidak akan diketahui oleh pihak lawan atau penyerang.

Reseeding dari *fast pool* menggunakan kunci yang sekarang dan *hash* dari semua *input* yang masuk kedalam *fast pool* sejak *reseed* terakhir dilakukan untuk membangkitkan kunci baru. Setelah hal ini selesai dilakukan, maka estimasi *entropy* untuk *fast pool* akan di-*reset* menjadi bernilai nol.

Reseeding dari *slow pool* menggunakan kunci yang sekarang dan *hash* dari semua *input* yang masuk kedalam *fast pool* dan juga *hash* dari semua *input* yang masuk ke dalam *slow pool* sejak *reseed* terakhir dilakukan juga untuk membangkitkan kunci baru. Setelah hal ini selesai dilakukan, maka estimasi *entropy* untuk *fast pool* dan *slow pool* akan di-*reset* menjadi bernilai nol.

3.2.4 *Reseed Control*

Mekanisme reseed control harus mempertimbangkan banyak hal. *Reseeding* dengan frekuensi tinggi seringkali lebih disukai, tetapi hal ini juga menyebabkan serangan berupa tebakan secara iteratif juga semakin sering.

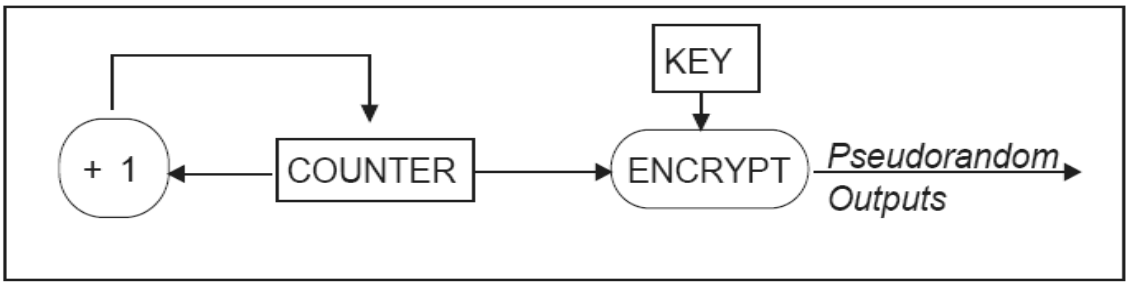


Figure 2 Mekanisme Pembangkitan Output

Namun *reseeding* dengan frekuensi rendah juga kurang menguntungkan. Perancangan komponen *reseed control* harus mempertimbangkan hal-hal tersebut.

Estimasi *entropy* untuk setiap sumber ketika *samples* masuk ke dalam *pool* harus terus dijaga. Ketika sumber di dalam *fast pool* sudah melewatu suatu nilai ambang batas, harus dilakukan *reseed* dari *fast pool*. Pada kebanyakan sistem, hal ini terjadi berulang kali dalam satu jam. Dan ketika sembarang k dari n sumber sudah melewati ambang batas yang lebih tinggi di dalam *slow pool*, maka dilakukan *reseed* dari *slow pool*. Proses ini pada umumnya jauh lebih lambat.

Untuk *Yarrow-160*, ambang batas untuk *fast pool* adalah 100 bit dan untuk *slow pool* adalah 160 bit. Paling sedikit, dua sumber yang berbeda harus melebihi 160 bit di dalam *slow pool* sebelum *slow pool* melakukan *reseed* (secara *default*).

3.3 Perancangan *Yarrow* Secara Umum dan *Yarrow-160*

Pada bagian ini, akan dijelaskan mengenai perancangan *Yarrow* secara umum dan juga *Yarrow-160*. Ini adalah deskripsi umum dengan menggunakan *cipher* blok dan fungsi *hash* sembarang. Jika kedua algoritma tersebut aman, dan *PRNG* mendapatkan *entropy* dalam jumlah yang cukup, maka *CSPRNG* yang dihasilkan akan kuat.

Seperti telah dijelaskan di atas, kita membutuhkan dua buah algoritma sebagai berikut:

1. Sebuah fungsi *hash* satu arah, $h(x)$ dengan pesan ringkas yang dihasilkan berukuran *m-bit*.
2. Sebuah *cipher* blok, $E()$ dengan kunci berukuran k-bit dan blok berukuran n-bit.

Fungsi *hash* yang digunakan diasumsikan memenuhi syarat-syarat berikut:

1. *Collision intractable*
2. Satu arah (*One-way*)
3. Diberikan sembarang M nilai-nilai *input*, maka nilai-nilai *output* yang dihasilkan tersebar sebagaimana $|M|$ pemilihan pada sebuah distribusi seragam dengan nilai-nilai m-bit.

Sedangkan *cipher* blok yang digunakan diasumsikan memenuhi syarat-syarat berikut:

1. Tahan terhadap serangan *known-plaintext* dan *chosen-plaintext*, artinya serangan-serangan tersebut membutuhkan plainteks yang sangat banyak beserta cipherteks pasangannya.
2. Memiliki sifat statistik keluaran yang bagus walaupun diberikan input dengan berbagai macam pola.

Secara khusus, *Yarrow-160* menggunakan *SHA-1* sebagai fungsi *h(x)* dan *triple-DES* sebagai fungsi E.

3.3.1 Mekanisme Pembangkitan *Output*

Gambar 2 menunjukkan pembangkit *output* dengan menggunakan *cipher* blok dalam mode *counter*.

Pertama, kita memiliki n-bit nilai *counter* C. Untuk membangkitkan n-bit blok *output*

berikutnya, kita *increment* nilai *C* kemudian mengenkripsikannya dengan *cipher* blok *E* dan kunci *K*. Jadi, untuk membangkitkan blok *output* berikutnya, kita melakukan hal berikut:

$$C \leftarrow (C+1) \bmod 2^n$$
$$R \leftarrow E_K(C)$$

Dimana *R* adalah blok *output* berikutnya dan *K* adalah kunci *PRNG* sekarang (*current PRNG key*)

3.3.2 *Entropy Accumulator*

Untuk mengumpulkan *entropy* dari serangkaian *input*, *input-input* tersebut disambung (*concatenation*). Setelah mendapatkan cukup *entropy*, dilakukan fungsi *hash* pada hasil penyambugan input tersebut. Hal ini dilakukan secara bergantian pada *samples* dari setiap sumber ke setiap *pool*.

Pada *Yarrow-160*, digunakan fungsi *hash SHA-1* untuk mengumpulkan input dengan cara tersebut.

3.3.3 *Reseed Mechanism*

Mekanisme *reseeding* membangkitkan kunci *K* baru yang digunakan oleh *cipher* blok pada komponen pembangkitan *output*. Kunci yang baru tersebut dibangkitkan dari kunci lama dan *entropy* di *pool*. Waktu eksekusi mekanisme *reseeding* ini tergantung pada suatu parameter $P_t \geq 0$. Parameter ini dapat berupa konstanta tetap atau dapat diatur secara dinamis.

Proses *reseed* terdiri dari langkah-langkah berikut:

1. *Entropy accumulator* menghitung nilai *hash* pada penyambungan semua input di dalam *fast pool*. Misalkan hasilnya v_0.
2. Definisikan $v_i = h(v_{i-1} | v_0 | i)$, untuk $i = 1,2,..., t$.
3. $K \leftarrow h'(h(v_P | K), k)$.
4. Definisikan $C \leftarrow E_K(0)$.
5. *Reset* semua estimasi *entropy* menjadi bernilai nol.
6. Hapus semua nilai-nilai perantara dari memori.
7. Jika sebuah *seed file* sedang digunakan, 2*k*-bit *output* berikutnya dari pembangkit dituliskan ke dalam *seed file* dan meng-*overwrite* nilai lama.

Dalam hal ini | merupakan operator penyambungan (*concatenation*), dan fungsi *h'* didefinisikan sebagai berikut: untuk menghitung *h'(m,k)*, kita konstruksi

$$s_0 = m$$
$$s_i = h(s_0 | ... | s_{i-1}),\ i = 1,2,...$$
$$h'(m,k) = \text{first } k \text{ bits of } (s_0 | s_1...)$$

3.3.4 *Reseed Control*

Modul *reseed control* digunakan untuk menentukan kapan proses *reseed* akan dilakukan. *Reseed* dapat terjadi secara eksplisit, yaitu ketika suatu aplikasi meminta operasi *reseed*. Namun hal ini sangat jarang dilakukan dan biasanya hanya dilakukan pada aplikasi yang membangkitkan bilangan acak yang nilainya sangat tinggi dan rahasia. Secara umum, akses untuk operasi *reseed* secara eksplisit seharusnya dibatasi.

Operasi *reseed* terjadi secara otomatis dan berulang secara periodik. *Fast pool* digunakan untuk *reseed* ketika salah satu sumbernya memiliki estimasi *entropy* di atas suatu nilai ambang batas. *Slow pool* digunakan untuk *reseed* ketika paling sedikit dua dari sumbernya memiliki estimasi *entropy* di atas suatu nilai ambang batas.

Pada *Yarrow-160*, nilai ambang batas untuk *fast pool* adalah 100 bit dan nilai ambang batas untuk *slow pool* adalah 160 bit.

4 Perbandingan *CSPRNG* *Blum-Blum Shub* dan *Yarrow*

Setelah melakukan studi dua *CSPRNG Blum Blum Shub (BBS)* dan *Yarrow*, sekarang kita akan melakukan perbandingan kedua *CSPRNG* tersebut. Perbandingan akan dilakukan dari segi algoritma, kerumitan komputasi, performansi dan keamanan bilangan-bilangan acak yang dihasilkan.

Dari segi algoritma, jelas bahwa *Blum Blum Shub* jauh lebih sederhana dibandingkan *Yarrow*. *Blum Blum Shub* hanya membutuhkan perhitungan pangkat dan *modulo* saja, sedangkan

Yarrow membutuhkan fungsi *hash* dan *cipher* blok tambahan. Selain itu, *Yarrow* juga memperhitungkan *entropy input*.

Lebih jauh lagi, jika nilai-nilai *p* dan *q* pada *Blum Blum Shub* diketahui, maka untuk menghitung bilangan acak ke-*i* yang dibangkitkan, tidak perlu melakukan iterasi, namun cukup dengan melakukan perhitungan berikut:

$$x_i = x_0^{2^i \bmod (p-1)(q-1)} \bmod n$$

Dengan algoritma yang jauh lebih sederhana, *Blum Blum Shub* memiliki tingkat kerumitan komputasi yang jauh lebih rendah dan performansi yang lebih baik dibandingkan *Yarrow*.

Namun dari segi keamanan, jelas *Yarrow* lebih aman dibandingkan *Blum Blum Shub*, walaupun keduanya sama-sama termasuk *cryptographically secure pseudo-random generators* (*CSPRNG*). Jaminan keamanan pada *Blum Blum Shub* ada karena sulitnya memecahkan masalah *quadratic residuosity* dan adanya beberapa asumsi-asumsi tambahan dalam teori bilangan. Ketika masalah tersebut dapat dipecahkan atau asumsi tersebut tidak berlaku lagi, maka jaminan keamanan pada *Blum Blum Shub* akan berkurang. Selain itu, nilai-nilai acak yang dibangkitkan oleh *Blum Blum Shub* hanya ditentukan oleh nilai *p,q,* dan *s*. Jika nilai-nilai *p,q,* atau *s* yang dipilih tidak bagus, ada kemungkinan terjadi perulangan bilangan-bilangan acak yang dibangkitkan dengan periode yang kecil. Hal ini jelas memudahkan penyerang untuk menebak bilangan acak yang dibangkitkan berikutnya.

Pada *Yarrow*, terdapat parameter-parameter tambahan lain sangat sulit diprediksi, yaitu kapan waktu yang dilakukan untuk melakukan *reseeding* dan adanya estimasi *entropy* yang dilakukan oleh komponen *entropy accumlator*. Selain itu digunakan fungsi *hash* satu arah dan *cipher* blok yang cukup sulit dipecahkan.

5 Kesimpulan

Kesimpulan yang dapat diambil dari studi dan perbandingan *CSPRNG Blum Blum Shub* dan *Yarrow* adalah:

1. Sejak jaman dahulu, manusia sudah menggunakan pembangkit bilangan acak. Dan sekarang, dalam aspek kriptografi, pembangkit bilangan acak juga memiliki fungsi yang sangat penting. Namun pembangkit bilangan acak yang benar-benar acak sangat sulit diimplementasikan, apalagi dengan menggunakan komputer yang bersifat *deterministik*. Oleh karena itu dilakukan pendekatan *Pseudo-random Number Generators* (*PRNG*) atau pembangkit bilangan acak semu. *PRNG* yang digunakan untuk tujuan kriptografi disebut *CSPRNG* atau *cryptographically secure pseudo-random number generators*. Dua *CSPRNG* yang cukup sering digunakan adalah *Blum Blum Shub* dan *Yarrow*. *Blum Blum Shub* digunakan karena sangat sederhana dan mangkus. Bilangan acak yang dihasilkan juga dapat dikatakan lumayan bagus karena melewati uji *CSPRNG*. *Yarrow* digunakan karena tingkat keamanan yang sangat tinggi, walaupun tidak semangkus *Blum Blum Shub*. Dengan berbagai parameter yang sangat sulit diprediksi, *Yarrow* dapat dikatakan sebagai *CSPRNG* yang mendekati *true random number generator*.

2. *CSPRNG Blum Blum Shub* merupakan *PRNG* yang sepenuhnya didasarkan pada teori bilangan. Dengan berbagai teorema dalam teori bilangan, dapat dibuktikan bahwa *CSPRNG Blum Blum Shub* termasuk *PRNG* yang aman untuk kriptografi walaupun sepintas algoritma *PRNG Blum Blum Shub* terlihat sangat sederhana, yaitu hanya dengan menggunakan perhitungan pangkat dan *modulo* saja.

3. *CSPRNG Yarrow* sekarang ini termasuk salah satu *CSPRNG* dengan tingkat keamanan paling tinggi. *Yarrow* terdiri dari empat komponen utama, yaitu *Entropy Accumulator, Generation Mechanism, Reseed Mechanism* dan *Reseed Control*. Selain itu, *Yarrow* juga membutuhkan dua algoritma tambahan yaitu fungsi *hash* dan *cipher* blok.

4. Kedua *CSPRNG Blum Blum Shub* dan *Yarrow* memiliki kelebihan dan kekurangan masing-masing. *Blum Blum Shub* unggul dalam hal kesederhanaan algoritma, kerumitan komputasi lebih sedikit dan performansi yang lebih baik. Di sisi lain, *Yarrow* memiliki tingkat keamanan yang lebih baik.

Daftar Pustaka

[1] Munir, Rinaldi. *Diktat Kuliah IF5054 Kriptografi*

[2] Schneier, Bruce. *Applied Cryptogrphy.*

[3] Schneier, Bruce. *Cryptanalytic Attacks on Pseudo-Random Number Generators*

[4] Ruggeri, Ned. *Principles of Pseudo Random Number Generators*

[5] Junod, Pascal. *Cryptographic Secure Pseudo-Random Bits Generation: The Blum Blum Shub Generator*

[6] Schenier, Bruce, et.al. *Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator*

[7] Jones, Gareth A. *Elementary Number Theory*

[8] Mundle, Maithily. *Various Implementation of Blum Blum Shub Pseudo Random Number Generator*

[9] *Murray, Mark R.V. An Implementation of the Yarrow PRNG for FreeBSD.*