Stinson textbook

Definition 8.1: Let k, ℓ be positive integers such that $\ell \geq k+1$. A *(k, ℓ)-bit generator* is a function $f : (\mathbb{Z}_2)^k \rightarrow (\mathbb{Z}_2)^\ell$ that can be computed in polynomial time (as a function of k). The input $s_0 \in (\mathbb{Z}_2)^k$ is called the *seed*, and the output $f(s_0) \in (\mathbb{Z}_2)^\ell$ is called the *generated bitstring*. It will always be required that ℓ is a polynomial function of k.

wanted : fast - simulations
uniform - Monte Carlo
unpredictable - keys

Algorithm 8.1: *Linear Congruential Generator*

Suppose $M \geq 2$ is an integer and suppose $1 \leq a, b \leq M - 1$. Define $k = 1 + \lfloor \log_2 M \rfloor$ and let $k + 1 \leq \ell \leq M - 1$.

The seed is an integer s_0, where $0 \leq s_0 \leq M - 1$. (Observe that the binary representation of a seed is a bitstring of length not exceeding k; however, not all bitstrings of length k are permissible seeds.) Now, define

$$s_i = (as_{i-1} + b) \bmod M$$

for $1 \leq i \leq \ell$, and then define

$$f(s_0) = (z_1, z_2, \ldots, z_\ell),$$

where

$$z_i = s_i \bmod 2,$$

$1 \leq i \leq \ell$. Then f is a (k, ℓ)-*Linear Congruential Generator.*

LFSR

merging LFSRs

fast/uniform

TABLE 8.1
Bitstrings produced by the linear congruential generator

seed	sequence
0	1010001101
1	0100110101
2	1101010001
3	0001101001
4	1100101101
5	0100011010
6	1000110010
7	0101000110
8	1001101010
9	1010011010
10	0110010110
11	1010100011
12	0011001011
13	1111111111
14	0011010011
15	1010100011

seed	sequence
16	0110100110
17	1001011010
18	0101101010
19	0101000110
20	1000110100
21	0100011001
22	1101001101
23	0001100101
24	1101010001
25	0010110101
26	1010001100
27	0110101000
28	1011010100
29	0011010100
30	0110101000

$a=3$
$b=5$

$M=31$
$5 \to 10$ bits

Algorithm 8.2: *RSA Generator*

Suppose p, q are two $(k/2)$-bit primes, and define $n = pq$. Suppose b is chosen such that $\gcd(b, \phi(n)) = 1$. As always, n and b are public while p and q are secret.

A seed s_0 is any element of $\mathbb{Z}_n{}^*$, so s_0 has k bits. For $i \geq 1$, define

$$s_{i+1} = s_i{}^b \bmod n,$$

and then define

$$f(s_0) = (z_1, z_2, \ldots, z_\ell),$$

where

$$z_i = s_i \bmod 2,$$

$1 \leq i \leq \ell$. Then f is a (k, ℓ)-*RSA Generator.*

slow
unpredictable

TABLE 8.2
Bits produced by the *RSA Generator*

i	s_i	z_i
0	75634	
1	31483	1
2	31238	0
3	51968	0
4	39796	0
5	28716	0
6	14089	1
7	5923	1
8	44891	1
9	62284	0
10	11889	1
11	43467	1
12	71215	1
13	10401	1
14	77444	0
15	56794	0
16	78147	1
17	72137	1
18	89592	0
19	29022	0
20	13356	0

$n = 91261$
$= 263 * 347$

$b = 1547$

$s_0 = 75634$

Algorithm 8.5: *Blum-Blum-Shub Generator*

Let p, q be two $(k/2)$-bit primes such that $p \equiv q \equiv 3 \bmod 4$, and define $n = pq$. Let $\mathbf{QR}(n)$ denote the set of quadratic residues modulo n.

A seed s_0 is any element of $\mathbf{QR}(n)$. For $0 \leq i \leq \ell - 1$, define

$$s_{i+1} = {s_i}^2 \bmod n,$$

and then define

$$f(s_0) = (z_1, z_2, \ldots, z_\ell),$$

where

$$z_i = s_i \bmod 2,$$

$1 \leq i \leq \ell$. Then f is a (k, ℓ)-PRBG, called the *Blum-Blum-Shub Generator*, which we abbreviate to *BBS Generator*.

One way to choose an appropriate seed is to select an element $s_{-1} \in {\mathbb{Z}_n}^*$ and compute $s_0 = {s_{-1}}^2 \bmod n$. This ensures that $s_0 \in \mathbf{QR}(n)$.

Some claim: can use loglog n bits from each iteration

Algorithm 8.8: *Discrete Logarithm Generator*

Let p be a k-bit prime, and let α be a primitive element modulo p.

A seed x_0 is any element of $\mathbb{Z}_p{}^*$. For $i \geq 0$, define

$$x_{i+1} = \alpha^{x_i} \bmod p,$$

and then define

$$f(x_0) = (z_1, z_2, \ldots, z_\ell),$$

where

$$z_i = \begin{cases} 1 & \text{if } x_i > p/2 \\ 0 & \text{if } x_i < p/2. \end{cases}$$

Then f is called a (k, ℓ)-*Discrete Logarithm Generator.*

CRC Handbook of Applied Crypto

5.11 **Algorithm** ANSI X9.17 pseudorandom bit generator

INPUT: a random (and secret) 64-bit seed s, integer m, and DES E-D-E encryption key k.
OUTPUT: m pseudorandom 64-bit strings $x_1, x_2, \ldots, x_m$.

1. Compute the intermediate value $I = E_k(D)$, where D is a 64-bit representation of the date/time to as fine a resolution as is available.
2. For i from 1 to m do the following:
   2.1 $x_i \leftarrow E_k(I \oplus s)$.
   2.2 $s \leftarrow E_k(x_i \oplus I)$.
3. Return$(x_1, x_2, \ldots, x_m)$.

5.15 **Algorithm** FIPS 186 one-way function using SHA-1

INPUT: a 160-bit string t and a b-bit string c, $160 \leq b \leq 512$.
OUTPUT: a 160-bit string denoted $G(t, c)$.

1. Break up t into five 32-bit blocks: $t = H_1 \| H_2 \| H_3 \| H_4 \| H_5$.
2. Pad c with 0's to obtain a 512-bit message block: $X \leftarrow c \| 0^{512-b}$.
3. Divide X into 16 32-bit words: $x_0 x_1 \ldots x_{15}$, and set $m \leftarrow 1$.
4. Execute step 4 of SHA-1 (Algorithm 9.53). (This alters the H_i's.)
5. The output is the concatenation: $G(t, c) = H_1 \| H_2 \| H_3 \| H_4 \| H_5$.

5.16 Algorithm FIPS 186 one-way function using DES

INPUT: two 160-bit strings t and c.

OUTPUT: a 160-bit string denoted $G(t, c)$.

1. Break up t into five 32-bit blocks: $t = t_0 \| t_1 \| t_2 \| t_3 \| t_4$.
2. Break up c into five 32-bit blocks: $c = c_0 \| c_1 \| c_2 \| c_3 \| c_4$.
3. For i from 0 to 4 do the following: $x_i \leftarrow t_i \oplus c_i$.
4. For i from 0 to 4 do the following:
   - 4.1 $b_1 \leftarrow c_{(i+4) \bmod 5}$, $b_2 \leftarrow c_{(i+3) \bmod 5}$.
   - 4.2 $a_1 \leftarrow x_i$, $a_2 \leftarrow x_{(i+1) \bmod 5} \oplus x_{(i+4) \bmod 5}$.
   - 4.3 $A \leftarrow a_1 \| a_2$, $B \leftarrow b_1' \| b_2$, where b_1' denotes the 24 least significant bits of b_1.
   - 4.4 Use DES with key B to encrypt A: $y_i \leftarrow \mathrm{DES}_B(A)$.
   - 4.5 Break up y_i into two 32-bit blocks: $y_i = L_i \| R_i$.
5. For i from 0 to 4 do the following: $z_i \leftarrow L_i \oplus R_{(i+2) \bmod 5} \oplus L_{(i+3) \bmod 5}$.
6. The output is the concatenation: $G(t, c) = z_0 \| z_1 \| z_2 \| z_3 \| z_4$.

5.12 Algorithm FIPS 186 pseudorandom number generator for DSA private keys

INPUT: an integer m and a 160-bit prime number q.
OUTPUT: m pseudorandom numbers $a_1, a_2, \ldots, a_m$ in the interval $[0, q-1]$ which may be used as DSA private keys.

1. If Algorithm 5.15 is to be used in step 4.3 then select an arbitrary integer b, $160 \le b \le 512$; if Algorithm 5.16 is to be used then set $b \leftarrow 160$.
2. Generate a random (and secret) b-bit seed s.
3. Define the 160-bit string t = 67452301 efcdab89 98badcfe 10325476 c3d2e1f0 (in hexadecimal).
4. For i from 1 to m do the following:
   - 4.1 (optional user input) Either select a b-bit string y_i, or set $y_i \leftarrow 0$.
   - 4.2 $z_i \leftarrow (s + y_i) \bmod 2^b$.
   - 4.3 $a_i \leftarrow G(t, z_i) \bmod q$. ($G$ is either that defined in Algorithm 5.15 or 5.16.)
   - 4.4 $s \leftarrow (1 + s + a_i) \bmod 2^b$.
5. Return$(a_1, a_2, \ldots, a_m)$.

5.14 Algorithm FIPS 186 pseudorandom number generator for DSA per-message secrets

INPUT: an integer m and a 160-bit prime number q.
OUTPUT: m pseudorandom numbers $k_1, k_2, \ldots, k_m$ in the interval $[0, q-1]$ which may be used as the per-message secret numbers k in the DSA.

1. If Algorithm 5.15 is to be used in step 4.1 then select an integer b, $160 \le b \le 512$; if Algorithm 5.16 is to be used then set $b \leftarrow 160$.
2. Generate a random (and secret) b-bit seed s.
3. Define the 160-bit string t = efcdab89 98badcfe 10325476 c3d2e1f0 67452301 (in hexadecimal).
4. For i from 1 to m do the following:
   - 4.1 $k_i \leftarrow G(t, s) \bmod q$. ($G$ is either that defined in Algorithm 5.15 or 5.16.)
   - 4.2 $s \leftarrow (1 + s + k_i) \bmod 2^b$.
5. Return$(k_1, k_2, \ldots, k_m)$.

5.35 Algorithm RSA pseudorandom bit generator

SUMMARY: a pseudorandom bit sequence $z_1, z_2, \dots, z_l$ of length l is generated.

1. *Setup*. Generate two secret RSA-like primes p and q (cf. Note 8.8), and compute $n = pq$ and $\phi = (p-1)(q-1)$. Select a random integer e, $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
2. Select a random integer x_0 (the *seed*) in the interval $[1, n-1]$.
3. For i from 1 to l do the following:
   - 3.1 $x_i \leftarrow x_{i-1}^e \bmod n$.
   - 3.2 $z_i \leftarrow$ the least significant bit of x_i.
4. The output sequence is $z_1, z_2, \dots, z_l$.

5.37 Algorithm Micali-Schnorr pseudorandom bit generator

SUMMARY: a pseudorandom bit sequence is generated.

1. *Setup*. Generate two secret RSA-like primes p and q (cf. Note 8.8), and compute $n = pq$ and $\phi = (p-1)(q-1)$. Let $N = \lfloor \lg n \rfloor + 1$ (the bitlength of n). Select an integer e, $1 < e < \phi$, such that $\gcd(e, \phi) = 1$ and $80e \leq N$. Let $k = \lfloor N(1 - \frac{2}{e}) \rfloor$ and $r = N - k$.
2. Select a random sequence x_0 (the *seed*) of bitlength r.
3. *Generate a pseudorandom sequence of length* $k \cdot l$. For i from 1 to l do the following:
   3.1 $y_i \leftarrow x_{i-1}^e \bmod n$.
   3.2 $x_i \leftarrow$ the r most significant bits of y_i.
   3.3 $z_i \leftarrow$ the k least significant bits of y_i.
4. The output sequence is $z_1 \parallel z_2 \parallel \cdots \parallel z_l$, where $\parallel$ denotes concatenation.

5.40 Algorithm Blum-Blum-Shub pseudorandom bit generator

SUMMARY: a pseudorandom bit sequence $z_1, z_2, \ldots, z_l$ of length l is generated.

1. *Setup*. Generate two large secret random (and distinct) primes p and q (cf. Note 8.8), each congruent to 3 modulo 4, and compute $n = pq$.
2. Select a random integer s (the *seed*) in the interval $[1, n-1]$ such that $\gcd(s, n) = 1$, and compute $x_0 \leftarrow s^2 \bmod n$.
3. For i from 1 to l do the following:
   3.1 $x_i \leftarrow x_{i-1}^2 \bmod n$.
   3.2 $z_i \leftarrow$ the least significant bit of x_i.
4. The output sequence is $z_1, z_2, \ldots, z_l$.