8. Random and Prime Numbers

8.1 Random number generator

8.1.1 True Random Number Generators

Device that generates random numbers from a physical process. Such devices are often based on microscopic phenomena. In theory, completely unpredictable. E.g.:coin flipping.

8.1.2 Pseudorandom Number Generator

Algorithm that approximate TRNG. It's not true, it is determined by a seed (initial value).

$$s_0=\text{seed} \quad s_{i+1}=f(s_i)$$

not always suitable for cryptography.

Linear Congruential Generator (LCG)

s0=seed   si+1=a*si+b mod m

Mersenne Twister PRNG

It is based on a "generalized" LFSR, i.e., a "twisted" generalized feedback shift register
It is used in many software/languages. It is not a good for cryptography.

8.1.3 Cryptographically Secure Pseudorandom Number Generators

CSPRNG is PRNG which is unpredictable: given a stream of n numbers generated by the CSPRNG is computationally infeasible to compute the subsequent numbers.

Implementations based on:

- cryptographic primitives, e.g., hash functions or ciphers.
- mathematical problems, e.g., Blum Blum Shub exploits the quadratic residuosity problem.
- special designs, e.g., Yarrow (/dev/random in Mac OS / iOS) evaluates quality of the inputs.

-Requirements:

Same requirement of PRNG plus:

- next-bit test: there is no polynomial-time algorithm that can predict the (k+1)th bit with $p > 50\%$

- withstand "state compromise extensions": in the event that part or all of its state has been revealed, it should be impossible to reconstruct the stream of random numbers prior to the revelation. It's should also be impossible to predict future state with entropy input.

-Netscape 1.1: Implementation with MD5, pid and ppid, problems:

1. MD5 is broken.

2. pid and ppid are 15 bits.

3. pid+(ppid<<12) gives only 27 bits of randomness.

4. pid can be leaked.

-Debian bug: predictable random number generator in openssl. They removed two lines of code: MD_Update(&m, buf, j);...MD_Update(&m, buf, j); it made the key generation phase predicatable.

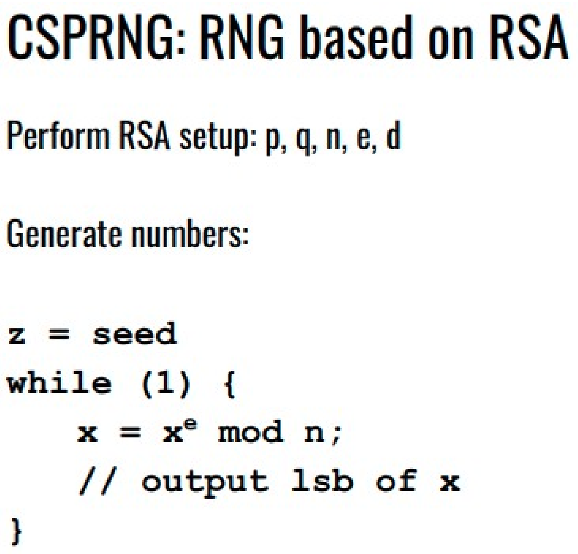
-BSI evaluation criteria

1. A sequence of random numbers with a low probability of containing identical consecutive elements (runs).

2. Several tests (monobit, poker etc.).

3. it should be impossible for attacker to calculate/guess from any sub-sequence.

4. It should be impossible for attacker to calculate/guess from an inner state.

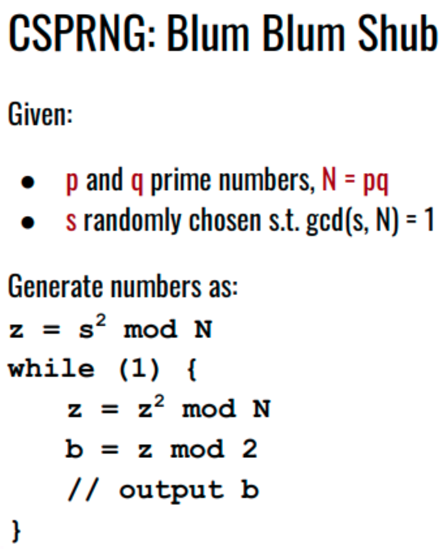


CSPRNG: RNG based on RSA

Perform RSA setup: p, q, n, e, d

Generate numbers:

```
z = seed
while (1) {
    x = $x^e$ mod n;
    // output lsb of x
}
```



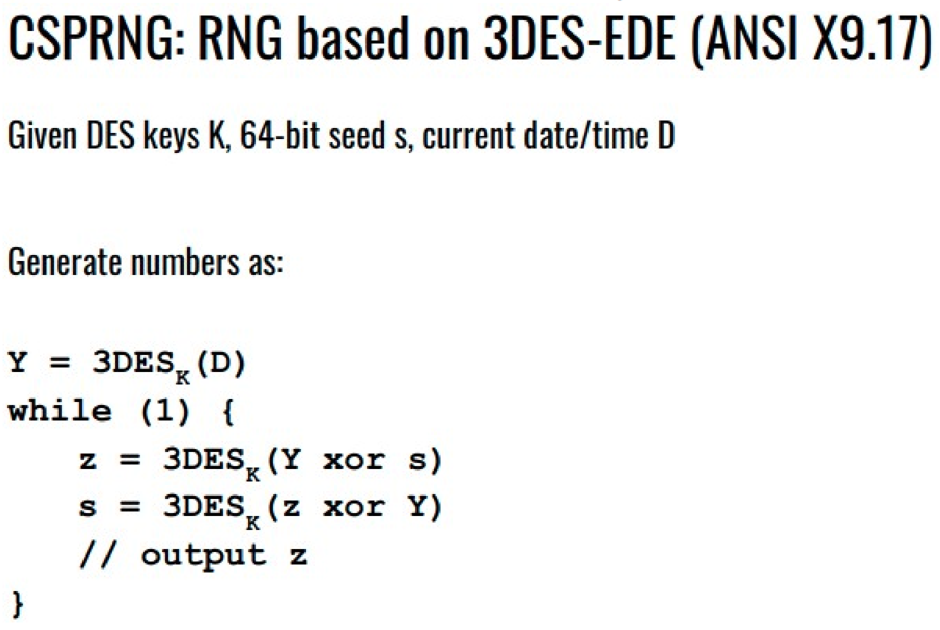
CSPRNG: Blum Blum Shub

Given:

- p and q prime numbers, N = pq
- s randomly chosen s.t. gcd(s, N) = 1

Generate numbers as:

```
z = $s^2$ mod N
while (1) {
    z = $z^2$ mod N
    b = z mod 2
    // output b
}
```



CSPRNG: RNG based on 3DES-EDE (ANSI X9.17)

Given DES keys K, 64-bit seed s, current date/time D

Generate numbers as:

```
Y = $3DES_K$(D)
while (1) {
    z = $3DES_K$(Y xor s)
    s = $3DES_K$(z xor Y)
    // output z
}
```

-How to choose seed?

With TRNG or combining unpredictable sources e.g.:

8.2 Kernel entropy pool

As the pool’s entropy diminishes (some numbers have been taken) as random numbers are handed out, the pool must be replenished: this process is called *stirring* (use of many events hard to predict, don’t trust TRNG).

On linux: > *cat /proc/sys/kernel/random/entropy_avail*

Use cryptography function to get new pseudo-random numbers; this neduce entropy, so use /dev/random to block reducing entropy till is “good enough”.

To find large prime numbers we use primality **probabilistic tests**;

E.g., if we want a prime number with 512 bits

$$P(\tilde{p} \text{ is prime}) \approx \frac{2}{\ln(2^{512})} = \frac{2}{512\ln(2)} \approx \frac{1}{177}$$

or with **Fermat Primality Test**

Algorithm:

```
1      FOR $i = 1$ TO $s$
1.1        choose random $a \in \{2,3,\ldots,\tilde{p}-2\}$
1.2        IF $a^{\tilde{p}-1} \not\equiv 1$
1.3            RETURN (“$\tilde{p}$ is composite”)
2      RETURN (“$\tilde{p}$ is likely prime”)
```

Anyway, it can be that $a^{p-1}=1$ but not prime, in fact it is not used in this form.

- Euclid’s Lemma

If a prime p divides a*b, the p must divide a or b.

8.3 Miller-Rabin Primality Test

Theorem 7.6.1 *Given the decomposition of an odd prime candidate $\tilde{p}$*

$$\tilde{p} - 1 = 2^u r$$

where r is odd. If we can find an integer a such that

$$a^r \not\equiv 1 \bmod \tilde{p} \qquad \text{and} \qquad a^{r2^j} \not\equiv \tilde{p}-1 \bmod \tilde{p}$$

for all $j = \{0,1,\ldots,u-1\}$, then $\tilde{p}$ is composite. Otherwise, it is probably a prime.

it can be proved, that using this decomposition strategy, we get the correct result only 3 out 4 times, i.e., 75%. As in the Fermat’s Primality test, we can try different values of a:

we test one value of a then ¼ to be wrong, two values of then 1/16 to wrong and so on.