Pseudo-Random Number Generators

Functional Programming and Intelligent Algorithms

Prof Hans Georg Schaathun

Høgskolen i Ålesund

14th February 2017

Randomness

1. What is randomness?

Randomness

1. What is randomness?
2. How do we create probabilistic computer programs?

Randomness

1. What is randomness?
2. How do we create probabilistic computer programs?
3. I.e. how do we make the computer act at random?

Two options

Two options

True randomness uses physical sources of entropy

1. `/dev/random` on many systems
2. `random-fu` in Haskell

Two options

True randomness uses physical sources of entropy

1. `/dev/random` on many systems
2. `random-fu` in Haskell

Pseudo-random number generators (PRNG) are deterministic but random-*looking*

- `random`, standard package in Haskell
- `random-tf`, more recent Haskell package

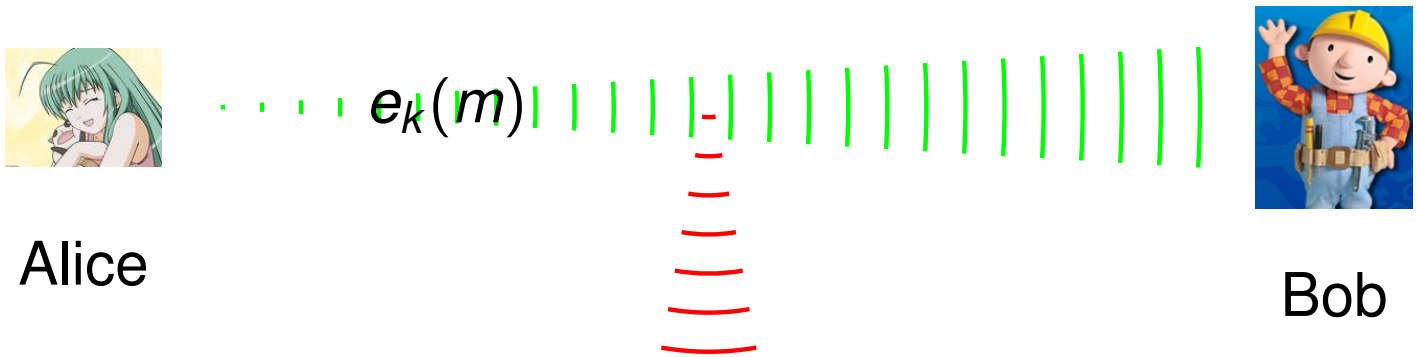
Linear Congruential Generators

$$x_i = a + cx_{i-1} \mod m,$$
$$x_0 \text{ is a given seed}$$

— Pseudo-random sequence $[x_0, x_1, x_2, \ldots]$

— Aka. *Lehmer's algorithm*

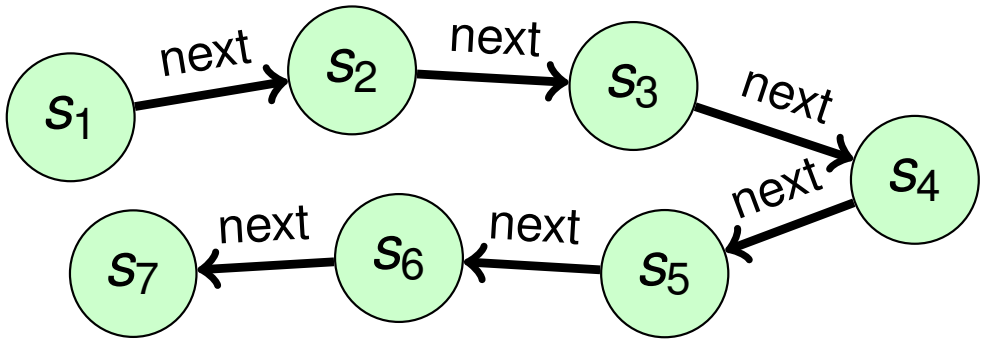
Ciphers in counter mode



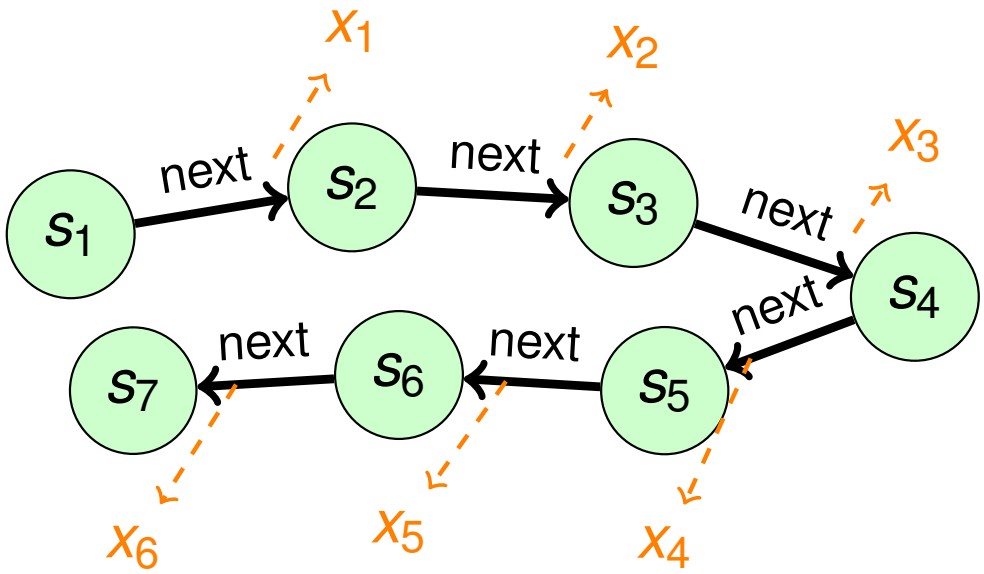
$x_i = e_k(i)$
x_0 is a given seed

Pseudo-random sequence
$[x_0, x_1, x_2, \ldots]$

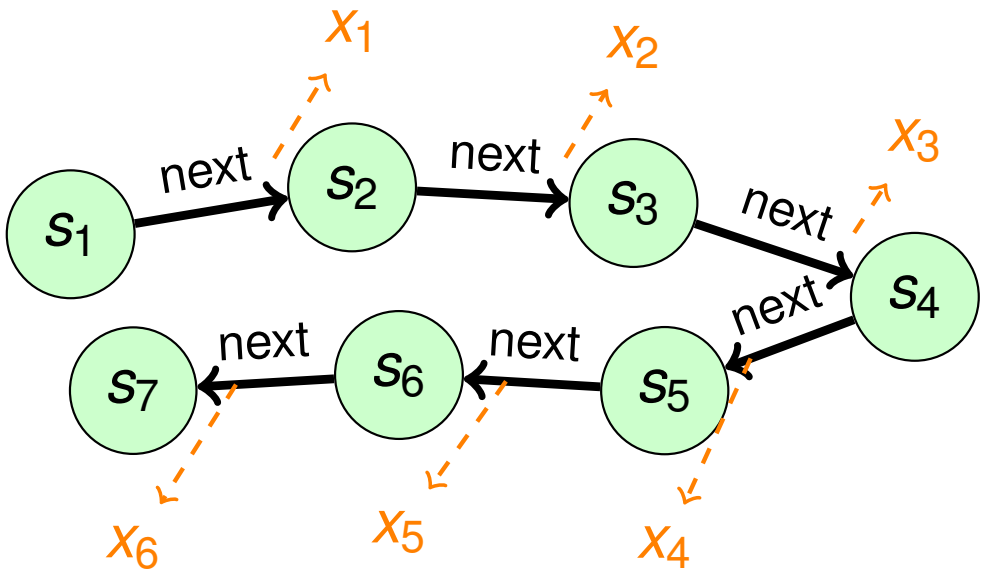
The PRNG is a state machine



The PRNG is a state machine

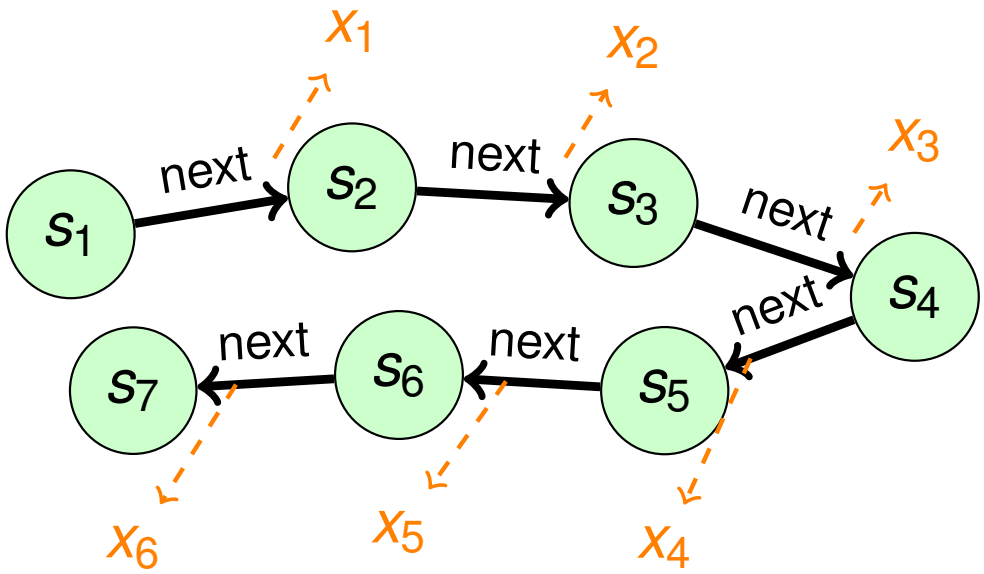


The PRNG is a state machine




```
— next ::  State -> (State,Int)
```

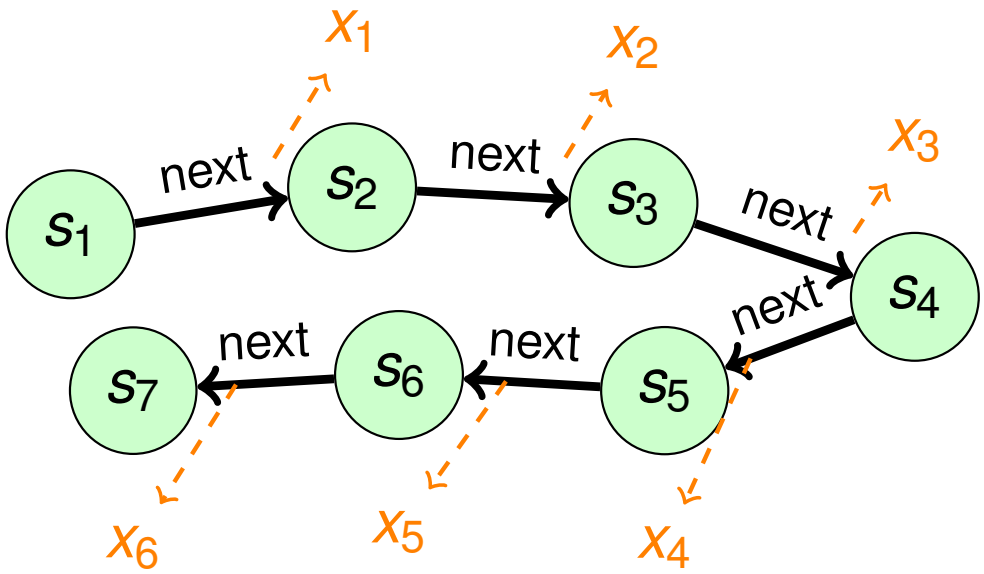
The PRNG is a state machine



- `next :: State -> (State,Int)`
- Lehmer:

```
next s = (s',s')
    where s' = (a + x*s) `mod` m
```

The PRNG is a state machine



- `next :: State -> (State,Int)`
- Lehmer:
```
next s = (s',s')
    where s' = (a + x*s) `mod` m
```
- Cipher: `next s = (s + 1 `mod` m, encrypt k s)`

`random-tf` package

1. `next :: TFGen -> (TFGen,Word32)`

Exercise

Given a TFGen object, how do you generate an random, infinite list of Word32 objects?

Splitting a PRNG

1. `split :: TFGen -> (TFGen,TFGen)`
2. `(g',newstate) = split g`
3. Use `g'` to generate the list
4. `newstate` is your new state

Where do you get the initial state?

Where do you get the initial state?

1. Hardcode an arbitrary seed
2. Use initialisation functions in the library
   2.1 `initTFGen`
3. Use a library which provides true random values
   - random-fu

Tuning parameters

1. Distribution of random initial weights?
2. β in the sigmoid function?
3. Number of iterations?

Some guidelines

— Weights: $-1/\sqrt{n} \leq w \leq 1/\sqrt{n}$
  - where n is the number of inputs to the layer

— The weights should have similar magnitude

— Small β — $\beta \leq 3$
  1. $\beta = 1$ is a good starting point

Number of epochs

Exercise

- Random starting weights
  1. `initNeuron`
  2. `initNetwork`
- Test your network
- Experiment by varying
  1. magnitude of initial weights
  2. β
  3. number of epochs