

The Fortuna PRNG

Niels Ferguson



1

The problem

- We need to make “random” choices in cryptographic protocols.
- Computers are deterministic.
- Standard “random” functions are completely inadequate.
- Must be unpredictable for an attacker!

2

“Real” RNG

- Random Number Generator.
- A “real” RNG is a specialised hardware device that produces random numbers.
- Difficult and expensive.
- Prone to undetected failures.
- Not included in my PC.

3

PRNG

- Pseudo-Random Number Generator.
- Deterministic algorithm.
- Produces “random-looking” numbers.
- Widely used.
- Most of them cryptographically useless.

4

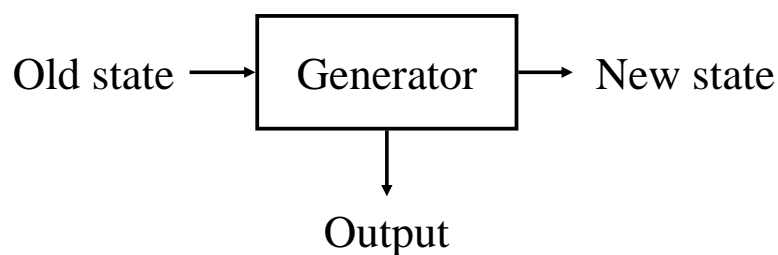
CSPRNG

- Cryptographically Strong Pseudo-Random Number Generator.
- Deterministic algorithm.
- Can't predict one output given other output values.
- Don't ever use anything else.

5

PRNG internals

- Internal state.
- State initialised with a random *seed*.
- Generator function that produces some output bits plus a new state.



6

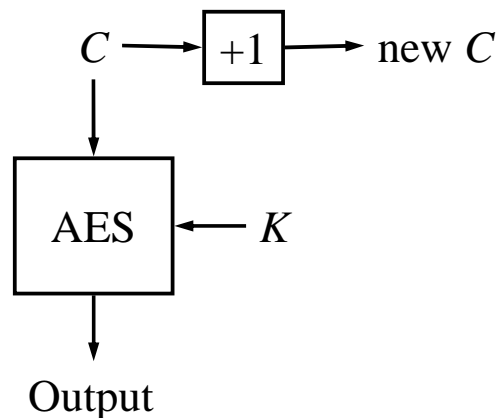
Generator function

- Can be built from existing cryptographic primitives:
- Block ciphers.
- Hash functions.
- Stream ciphers.

7

Fortuna generator

- State consists of 256-bit key K + 128-bit counter C .



8

State compromise

- What happens if an attacker manages to break into the machine and retrieve the PRNG state?
- This reveals all past outputs!
- It also reveals all future outputs.

9

Forward security

- Generate as many output blocks as required for the request.
- Generate two more blocks.
- Wipe key K .
- Set K to the two extra generated blocks.
- Protects against future state compromise.

10

Initialising the generator

- Set $(C, K) = (0, 0)$.
- Refuse to generate any output if $C = 0$.
- (Re)seed operation is required for first output.

11

(Re)seeding the generator

- Add seed material to the existing state.
- $K = \text{SHA}_d\text{-256}(K \parallel \text{seed string})$
- $C = C + 1$

12

The big problem

Where does the seed come from?

13

Solution 1

- Let the application choose the seed.

14

Problem 1

- This is not a solution.
- It just moves the problem to someone else, who is less well equipped to handle it.

15

Solution 2

- Use the current time as the seed.

16

Problem 2

- This is not a secret.
- Attacker can guess the time value.
- Guess can be verified by looking at *some* random data.
- Correct guess reveals *all* random data.

17

Solution 3

- Construct the seed from the current time, the process ID, the value of the windows handle, etc.

18

Problem 3

- Still not a secret.
- Requires a bit more guessing, but not a whole lot.

19

Seed requirements

- Seed must be unpredictable to the attacker.
- Measure of unpredictability: *entropy*

20

Entropy

- Unit of information.
- Formal definitions exist.
- A bit that will be set with probability 0.5 has 1 bit of entropy.
- A bit that will be set with probability 0.75 has 0.811... bits of entropy.
- A bit that will be set with probability 0.9 has 0.469... bits of entropy

21

Entropy (continued)

- Entropy of a variable depends on how many possible values it can have.
- Entropy depends on the probability distribution of those values.
- Entropy *also* depends on the knowledge we already have.
- Entropy is always *relative to someone's knowledge*.

22

White lie

- There are several different kinds of entropy.
- Shannon entropy: from information theory.
- Renyi entropy: used to analyse probability of duplicate values.
- Guessing entropy: most appropriate here.
- I will ignore these complications.

23

Back to reseeding

- Seed must have high entropy *with respect to the knowledge that the attacker has*.
- We don't know how much the attacker knows.
- We can't know how much effective entropy any particular value has.

24

Solution 4

- Ask the user to type some random text.

25

Problem 4

- Low entropy: many users produce something like “fdsajkl;fdsjkl;fsdajkl;fsdaj”
- Attacker can try to stuff keystrokes into the keyboard buffer.
- Or maybe she can sniff the keystrokes in some way.

26

Fundamental problem

- Entropy is not available in sufficient quantities when we want it.
- Initial reseed must come from external source.
- Let's leave this problem for now.

27

Recovery from compromise

- Assume that attacker knows our PRNG state.
- Irregular stream of “random events”.
- Can we evict the attacker and recover to an unknown state?

28

Typical event sources

- Mouse movements & detailed timings.
- Keyboard data & timings.
- Disk response speed.
- Clock jitter.
- Network packet timing.
- ...

29

Assumption

- Together, the events contain at least some entropy w.r.t. the attacker.
- Otherwise there is no hope of recovery.

30

Reseed on every event

- Simple solution.
- Reseed sets $K = \text{SHA}_d\text{-256}(K \parallel \text{event data})$.
- Harmless to reseed with non-random event.

31

Problem

- Does not work.
- Attacker knows old state, and can guess the event data.
- Asks for some random data from generator, and verify the guess.

32

Pooling

- Save up events in a pool.
- Reseed with whole pool at once.
- Attacker has to guess the whole pool.
- Recovers from state compromise if entropy of pool is large enough.

33

How large a pool

- Should pool events until we have, say, 128 bits of entropy.
- But we don't know the entropy of the events.
- Can't decide how many events to pool.
- Need to estimate the entropy.

34

Entropy estimators

- Don't even try.
- Impossible: depends on knowledge of the attacker.
- Various estimator systems have been proposed.
- All are ad-hoc, and heuristic.

35

Fortuna pooling

- Keep 32 pools: P_0, \dots, P_{31} .
- Each event source distributes its events over the pools.
- Entropy flows into pools at approximately the same rate.
- But we don't know at which rate.

36

Fortuna reseeding

- Reseed whenever P_0 is large enough to be interesting.
- Include pool P_k every 2^k reseeds.

37

Heuristic analysis

- T = Time between reseeds.
- R = rate at which entropy flows into each pool.
- Pool k is used in a reseed every $T \cdot 2^k$ seconds.
- In that time it has collected $R \cdot T \cdot 2^k$ bits of entropy.

38

One “good” pool

- As long as R and T are reasonable, there is at least one pool that collects 128 bits of entropy before it is used.
- We don't know which pool.
- But we don't care!
- Within a factor of 64 of optimal.

39

Further tricks

- Require that $T \geq 0.1$ s.
- Don't store the pools, but hash the pools.

40

Back to the initial seed

- We still need to have a starting seed value.

41

Seed file

- “Random” data on persistent storage.
- Read at startup and use to seed the PRNG.
- Danger: using the same seed file twice.
- Danger: compromise of the seed file.

42

At startup

- Read the seed file.
- Use contents to seed the PRNG.
- Generate new contents for seed file using PRNG.
- Write new seed file.
- Wait until write is permanent. (How?)
- Allow PRNG to be used.

43

Regularly

- Write new seed file.
- This incorporates the events that were used during the reseeds into the seed file state.

44

On shutdown

- Reseed with all the data from all the pools.
- Write a new seed file.
- Beware: losing the seed file is far worse than not writing a new one.
- Maybe use two files?

45

Initial seed file

- Unsolved problem.
- We need an external source for the seed file.
- Require factory to provide one?
- Ask user help during installation of OS?

46

Don't ignore the problem

- The first thing you (should) do on a new computer is to generate cryptographic keys for various purposes.
- Those have to be secure.
- This needs a good PRNG... which requires a good initial seed file.

47

Seed file details

- Closely look at the atomicity of file updates.
- Most operating systems don't promise anything.
- Even disk drives might buffer writes.
- Do the best you can.

48

Three parts of Fortuna

- Generator to service the requests.
- Pools & reseed control to recover from state compromise.
- Seed file & logic to provide starting seed.
- Required: initial seed file.

49

Where?

- Operating system has access to all events that can be used in a PRNG.
- Therefore, PRNG should be part of the operating system.
- It isn't in most cases.
- Work to be done!

50

Hardware RNG

- Don't use it directly.
- Use it as an event source for Fortuna.
- Keeps you safe if your RNG breaks (which happens more often than you'd like).
- Can use it for initialisation.

51

Warning

- Some systems have “real random” RNGs.
- Try to extract entropy from the events.
- Have to use entropy estimators.
- Pure heuristics, and therefore dangerous.

52

Conclusion

- A good PRNG is vital for good cryptographic systems.
- Fortuna is the state of the art.
- Relatively easy to implement, except for the parts that are inherently hard.
- Don't settle for anything less.