WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Français
Italiano
Русский
Українська
Edit links

Fortuna (PRNG)

From Wikipedia, the free encyclopedia

This article includes a list of references, related reading or external links, **but its sources remain unclear because it lacks inline citations**. Please improve this article by introducing more precise citations. *(September 2013)*

This article **relies too much on references to primary sources**. Please improve this article by adding secondary or tertiary sources. *(September 2013)*

Fortuna is a cryptographically secure pseudorandom number generator (PRNG) devised by Bruce Schneier and Niels Ferguson and published in 2003. It is named after Fortuna, the Roman goddess of chance.

Contents [hide]

Design [edit]

Fortuna is a *family* of secure PRNGs; its design leaves some choices open to implementors. It is composed of the following pieces:

- The generator itself, which once seeded will produce an indefinite quantity of pseudo-random data.
- The entropy accumulator, which collects genuinely random data from various sources and uses it to reseed the generator when enough new randomness has arrived.
- The seed file, which stores enough state to enable the computer to start generating random numbers as soon as it has booted.

Generator [edit]

The generator is based on any good block cipher. *Practical Cryptography* suggests AES, Serpent or Twofish. The basic idea is to run the cipher in counter mode, encrypting successive values of an incrementing counter. On its own, this would produce statistically identifiable deviations from randomness; for instance, generating 2^{64} genuinely random 128-bit blocks would produce on average about one pair of identical blocks, but there are no repeated blocks at all among the first 2^{128} produced by a 128-bit cipher in counter mode. Therefore, the key is changed periodically: no more than 1 MiB of data is generated without a key change. The key is also changed after every data request (however small), so that a key compromise doesn't endanger old RNG outputs.

Entropy accumulator [edit]

The entropy accumulator is designed to be resistant against "injection" attacks, without needing sophisticated (and inevitably unreliable) estimators of entropy. There are several "pools" of entropy; each entropy source distributes its alleged entropy evenly over the pools; and (here is the key idea) on the nth reseeding of the generator, pool k is used only if 2^k divides n. Thus, the kth pool is used only $1/2^k$ of the time. Higher-numbered pools, in other words, (1) contribute to reseedings less frequently but (2) collect a larger amount of entropy between reseedings. Reseeding is performed by hashing the specified entropy pools into the block cipher's key using two iterations of SHA-256.

Seeding [edit]

Unless an attacker is able to control *all* the sources of alleged entropy flowing into the system (in which case no algorithm can save it from compromise), there will be some *k* for which the *k*th pool collects enough entropy between reseedings that a reseeding with that pool ensures security. And that pool will be used at an interval proportional to the amount of entropy in question. Therefore, the system will always recover from an injection attack, and the time it takes to do so is at most a constant factor greater than the theoretical time it could take if we were able to identify which sources of entropy were corrupt and which not.

This conclusion depends on there being enough pools. Fortuna uses 32 pools, and restricts reseeding to happen at most 10 times per second. Running out of pools would then take about 13 years, which Ferguson and Schneier deem long enough for practical purposes. More paranoid implementors, or ones requiring the generation of random data at a colossal rate and correspondingly frequent reseeding, could use a larger number of pools.

Alternatives [edit]

Fortuna differs from the earlier Yarrow algorithm family of Schneier, Kelsey and Ferguson mostly in its handling of the entropy accumulator. Yarrow required each source of entropy to be accompanied by a mechanism for estimating the actual entropy supplied, and used only two pools; and its suggested embodiment (called *Yarrow-160*) used SHA-1 rather than iterated SHA-256.

Analysis [edit]

An analysis of Fortuna—and a proposed improvement—can be found in Y. Dodis, A. Shamir, N. Stephens-Davidowitz, D. Wichs, How to Eat Your Entropy and Have it Too —Optimal Recovery Strategies for Compromised RNGs Cryptology ePrint Archive, Report 2014/167, 2014.

See also [edit]

- Blum Blum Shub
- CryptGenRandom
- Random number generator attack
- Yarrow algorithm

References [edit]

- Niels Ferguson and Bruce Schneier, *Practical Cryptography*, published by Wiley in 2003. ISBN 0-471-22357-3.
- John Viega, "Practical Random Number Generation in Software," acsac, pp. 129, 19th Annual Computer Security Applications Conference (ACSAC '03), 2003

External links [edit]

- "Javascript Crypto Library". *includes a Javascript implementation of Fortuna PRNG*.
- Cooke, Jean-Luc (2005). "jlcooke's explanation of and improvements on /dev/random". *Patch adding an implementation of Fortuna to the Linux kernel*.
- Litzenberger, Dwayne (2013-10-20). "Fortuna implementation in Python, part of the Python Cryptography Toolkit".
- "How to Eat Your Entropy and Have it Too -- Optimal Recovery Strategies for Compromised RNGs". 2014-03-14.
- "Fortuna implementation in C++11". *Includes example server, entropy sources and command-line client*. 2015-06-01.

v · t · e **Cryptography**

History of cryptography · Cryptanalysis · Cryptography portal · Outline of cryptography

Symmetric-key algorithm · Block cipher · Stream cipher · Public-key cryptography · Cryptographic hash function · Message authentication code · Random numbers · Steganography

Categories: Pseudorandom number generators
Cryptographically secure pseudorandom number generators

This page was last modified on 1 June 2015, at 18:23.