



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
[Deutsch](#)
[Español](#)
[فارسی](#)
[Italiano](#)
[עברית](#)
[日本語](#)
[Português](#)
[Русский](#)

 [Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)



Cryptographically secure pseudorandom number generator

From Wikipedia, the free encyclopedia
(Redirected from [Cryptographically secure pseudo-random number generator](#))

A **cryptographically secure pseudo-random number generator** (**CSPRNG**) or **cryptographic pseudo-random number generator** (**CPRNG**)^[1] is a [pseudo-random number generator](#) (PRNG) with properties that make it suitable for use in [cryptography](#).

Many aspects of cryptography require [random](#) numbers, for example:

- [key generation](#)
- [nonces](#)
- [one-time pads](#)
- [salts](#) in certain signature schemes, including [ECDSA](#), [RSASSA-PSS](#)

The "quality" of the randomness required for these applications varies. For example creating a [nonce](#) in some [protocols](#) needs only uniqueness. On the other hand, generation of a master [key](#) requires a higher quality, such as more [entropy](#). And in the case of [one-time pads](#), the [information-theoretic](#) guarantee of perfect secrecy only holds if the key material comes from a true random source with high entropy.

Ideally, the generation of random numbers in CSPRNGs uses entropy obtained from a high-quality source, which might be a [hardware random number generator](#) or perhaps unpredictable system processes — though unexpected correlations have been found in several such ostensibly independent processes. From an information-theoretic point of view, the amount of randomness, the entropy that can be generated, is equal to the entropy provided by the system. But sometimes, in practical situations, more random numbers are needed than there is entropy available. Also the processes to extract randomness from a running system are slow in actual practice. In such instances, a CSPRNG can sometimes be used. A CSPRNG can "stretch" the available entropy over more bits.

Contents

- 1 Requirements
- 2 Some background
- 3 Designs
 - 3.1 Designs based on cryptographic primitives
 - 3.2 Number theoretic designs
 - 3.3 Special designs
- 4 Standards
- 5 NSA backdoor in the Dual_EC_DRBG PRNG
- 6 References
- 7 External links

Requirements [\[edit\]](#)

The requirements of an ordinary PRNG are also satisfied by a cryptographically secure PRNG, but the reverse is not true. CSPRNG requirements fall into two groups: first, that they pass statistical randomness tests; and secondly, that they hold up well under serious attack, even when part of their initial or running state becomes available to an attacker.^[*citation needed*]

- Every CSPRNG should satisfy the [next-bit test](#). That is, given the first *k* bits of a random sequence, there is no polynomial-time algorithm that can predict the (*k*+1)th bit with probability of success better than 50%. [Andrew Yao](#) proved in 1982 that a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness.^[2]
- Every CSPRNG should withstand "state compromise extensions". In the event that part or all of its state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the revelation. Additionally, if there is an entropy input while running, it should be infeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state.

Example: If the CSPRNG under consideration produces output by computing bits of π in sequence, starting from some unknown point in the binary expansion, it may well satisfy the next-bit test and thus be statistically random, as π appears to be a random sequence. (This would be guaranteed if π is a [normal number](#), for example.) However, this algorithm is not cryptographically secure; an attacker who determines which bit of π (i.e. the state of the algorithm) is currently in use will be able to calculate all preceding bits as well.

Most PRNGs are not suitable for use as CSPRNGs and will fail on both counts. First, while most PRNGs outputs appear random to assorted statistical tests, they do not resist determined reverse engineering. Specialized statistical tests may be found specially tuned to such a PRNG that shows the random numbers not to be truly random. Second, for most PRNGs, when their state has been revealed, all past random numbers can be retrodicted, allowing an attacker to read all past messages, as well as future ones.

CSPRNGs are designed explicitly to resist this type of [cryptanalysis](#).

Some background [\[edit\]](#)

Santha and Vazirani proved that several bit streams with weak randomness can be combined to produce a higher-quality quasi-random bit stream.^[3] Even earlier, [John von Neumann](#) proved that a [simple algorithm](#) can remove a considerable amount of the bias in any bit stream^[4] which should be applied to each bit stream before using any variation of the Santha-Vazirani design. The field is termed [entropy extraction](#) and is the subject of active research (e.g., [N Nisan](#), [S Safra](#), [R Shaltiel](#), [A Ta-Shma](#), [C Umans](#), [D Zuckerman](#)).

Designs [\[edit\]](#)

In the discussion below, CSPRNG designs are divided into three classes: 1) those based on cryptographic primitives such as [ciphers](#) and [cryptographic hashes](#), 2) those based upon mathematical problems thought to be hard, and 3) special-purpose designs. The last often introduce additional entropy when available and, strictly speaking, are not "pure" pseudorandom number generators, as their output is not completely determined by their initial state. This addition can prevent attacks even if the initial state is compromised.

Designs based on cryptographic primitives [\[edit\]](#)

- A secure [block cipher](#) can be converted into a CSPRNG by running it in [counter mode](#). This is done by choosing a [random](#) key and encrypting a 0, then encrypting a 1, then encrypting a 2, etc. The counter can also be started at an arbitrary number other than zero. Assuming an n -bit block cipher the output can be distinguished from random data after around $2^{n/2}$ blocks since, following the [birthday problem](#), colliding blocks should become likely at that point, whereas a block cipher in CTR mode will never output identical blocks. For 64 bit block ciphers this limits the safe output size to a few gigabytes, with 128 bit blocks the limitation is large enough not to impact typical applications.
- A cryptographically secure [hash](#) of a counter might also act as a good CSPRNG in some cases. In this case, it is also necessary that the initial value of this counter is random and secret. However, there has been little study of these algorithms for use in this manner, and at least some authors warn against this use.^{[vague][5]}
- Most [stream ciphers](#) work by generating a pseudorandom stream of bits that are combined (almost always XORed) with the [plaintext](#); running the cipher on a counter will return a new pseudorandom stream, possibly with a longer period. The cipher can only be secure if the original stream is a good CSPRNG, although this is not necessarily the case (see the [RC4 cipher](#)). Again, the initial state must be kept secret.

Number theoretic designs [\[edit\]](#)

- The [Blum Blum Shub](#) algorithm has a security proof based on the difficulty of the [Quadratic residuosity problem](#). Since the only known way to solve that problem is to factor the modulus, it is generally regarded that the difficulty of [integer factorization](#) provides a conditional security proof for the Blum Blum Shub algorithm. However the algorithm is very inefficient and therefore impractical unless extreme security is needed.
- The [Blum-Micali algorithm](#) has an unconditional security proof based on the difficulty of the [discrete logarithm problem](#) but is also very inefficient.
- Daniel Brown of [Certicom](#) has written a 2006 security proof for [Dual_EC_DRBG](#), based on the assumed hardness of the [Decisional Diffie–Hellman assumption](#), the *x-logarithm problem*, and the *truncated point problem*. The 2006 proof explicitly assumes a lower *outlen* than in the Dual_EC_DRBG standard, and that the P and Q in the Dual_EC_DRBG standard (which were revealed in 2013 to be probably backdoored by NSA) are replaced with non-backdoored values.

Special designs [\[edit\]](#)


There are a number of practical PRNGs that have been designed to be cryptographically secure, including


- the [Yarrow algorithm](#) which attempts to evaluate the entropic quality of its inputs. Yarrow is used in [FreeBSD](#), [OpenBSD](#) and [Mac OS X](#) (also as [/dev/random](#)).
- the [Fortuna algorithm](#), the successor to Yarrow, which does not attempt to evaluate the entropic quality of its inputs.
- the function [CryptGenRandom](#) provided in [Microsoft's Cryptographic Application Programming Interface](#)
- [ISAAC](#) based on a variant of the [RC4 cipher](#)
- [Evolutionary algorithm](#) based on [NIST Statistical Test Suite](#).^{[6][7]}
- [arc4random](#)
- [AES-CTR](#) DRBG is often used as a random number generator in systems that use AES encryption.^{[8][9]}
- [ANSI X9.17 standard](#) (*Financial Institution Key Management (wholesale)*), which has been adopted as a [FIPS](#) standard as well. It takes as input a [TDEA \(keying option 2\)](#) key bundle k and (the initial value of) a 64 bit [random seed](#) s .^[10] Each time a random number is required it:
 - Obtains the current date/time D to the maximum resolution possible.
 - Computes a temporary value $t = \text{TDEA}_k(D)$
 - Computes the random value $x = \text{TDEA}_k(s \oplus t)$, where \oplus denotes bitwise [exclusive or](#).
 - Updates the seed $s = \text{TDEA}_k(x \oplus t)$

Obviously, the technique is easily generalized to any block cipher; [AES](#) has been suggested (Young and Yung, op cit, sect 3.5.1).


Standards [\[edit\]](#)

Several CSPRNGs have been standardized. For example,

- [FIPS 186-2](#) 
- [NIST SP 800-90A](#): This standard has three uncontroversial CSPRNGs named Hash_DRBG, HMAC_DRBG, and CTR_DRBG; and a PRNG named [Dual_EC_DRBG](#) which has been shown to not be cryptographically secure and probably has a [kleptographic](#) NSA backdoor.
- [ANSI X9.17-1985 Appendix C](#)
- [ANSI X9.31-1998 Appendix A.2.4](#)
- [ANSI X9.62-1998 Annex A.4](#), obsoleted by [ANSI X9.62-2005, Annex D \(HMAC_DRBG\)](#)

A good [reference](#)  is maintained by [NIST](#).

There are also standards for statistical testing of new CSPRNG designs:



- *A Statistical Test Suite for Random and Pseudorandom Number Generators*, [NIST Special Publication 800-22](#) .





NSA backdoor in the Dual_EC_DRBG PRNG [\[edit\]](#)

Main article: [Dual_EC_DRBG](#)

The Guardian and *The New York Times* have reported that the [National Security Agency](#) (NSA) inserted a PRNG into [NIST SP 800-90A](#) that had a [backdoor](#) which allows the NSA to readily decrypt material that was encrypted with the aid of [Dual_EC_DRBG](#). Both papers report^{[11][12]} that, as independent security experts long suspected,^[13] the NSA has been introducing weaknesses into CSPRNG standard 800-90; this being confirmed for the first time by one of the top secret documents leaked to the Guardian by [Edward Snowden](#). The NSA worked covertly to get its own version of the NIST draft security standard approved for worldwide use in 2006. The leaked document states that "eventually, NSA became the sole editor." In spite of the known potential for a backdoor and other known significant deficiencies with [Dual_EC_DRBG](#), several companies such as [RSA Security](#) continued using [Dual_EC_DRBG](#) until the backdoor was confirmed in 2013.^[14] [RSA Security](#) received a \$10 million payment from the NSA to do so.^[15]

References [\[edit\]](#)

- ↑ Huang, Andrew (2003). *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press Series. No Starch Press. p. 111. ISBN 9781593270292. Retrieved 2013-10-24. "[...] the keystream generator [...] can be thought of as a cryptographic pseudo-random number generator (CPRNG)."
- ↑ Andrew Chi-Chih Yao. *Theory and applications of trapdoor functions* . In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982.
- ↑ Miklos Santha. Umesh V. Vazirani (1984-10-24). *"Generating quasi-random sequences from slightly-random*

3. "Various Sampling Methods to Generate Pseudo-Random Sequences from Truly Random Sources"  (PDF). *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*. University of California. pp. 434–440. ISBN 0-8186-0591-X. Retrieved 2006-11-29.
4. [^] John von Neumann (1963-03-01). "Various techniques for use in connection with random digits". *The Collected Works of John von Neumann*. Pergamon Press. pp. 768–770. ISBN 0-08-009566-6.
5. [^] Adam Young, Moti Yung (2004-02-01). *Malicious Cryptography: Exposing Cryptovirology*[?]. sect 3.2: John Wiley & Sons. p. 416. ISBN 978-0-7645-4975-5.
6. [^] NIST. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" . NIST, Special Publication April 2010
7. [^] A. Poorghanad, A. Sadr, A. Kashanipour" Generating High Quality Pseudo Random Number Using Evolutionary Methods", IEEE Congress on Computational Intelligence and Security, vol. 9, pp. 331-335 , May,2008 [1] 
8. [^] David Kleidermacher, Mike Kleidermacher. "Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development" [?]. Elsevier, 2012. p. 256.
9. [^] George Cox, Charles Dike, and DJ Johnston. "Intel's Digital Random Number Generator (DRNG)" . 2011.
10. [^] *Handbook of Applied Cryptography* [?], Alfred Menezes, Paul van Oorschot, and Scott Vanstone, CRC Press, 1996, Chapter 5 Pseudorandom Bits and Sequences  (PDF)
11. [^] James Borger; Glenn Greenwald (6 September 2013). "Revealed: how US and UK spy agencies defeat internet privacy and security" [?]. *The Guardian*. The Guardian. Retrieved 7 September 2013.
12. [^] Nicole Perlroth (5 September 2013). "N.S.A. Able to Foil Basic Safeguards of Privacy on Web" [?]. The New York Times. Retrieved 7 September 2013.
13. [^] Bruce Schneier (15 November 2007). "Did NSA Put a Secret Backdoor in New Encryption Standard?" [?]. *Wired*. Retrieved 7 September 2013.
14. [^] Matthew Green. "RSA warns developers not to use RSA products" [?].
15. [^] Joseph Menn (20 December 2013). "Exclusive: Secret contract tied NSA and security industry pioneer" [?]. *Reuters*.

External links [[edit](#)]

- [RFC 4086](#), Randomness Requirements for Security
- [Java "entropy pool" for cryptographically secure unpredictable random numbers.](#)
- [Java standard class providing a cryptographically strong pseudo-random number generator \(PRNG\).](#)
- [Cryptographically Secure Random number on Windows without using CryptoAPI](#)
- [Conjectured Security of the ANSI-NIST Elliptic Curve RNG](#), Daniel R. L. Brown, IACR ePrint 2006/117.
- [A Security Analysis of the NIST SP 800-90 Elliptic Curve Random Number Generator](#), Daniel R. L. Brown and Kristian Gjøsteen, IACR ePrint 2007/048. To appear in CRYPTO 2007.
- [Cryptanalysis of the Dual Elliptic Curve Pseudorandom Generator](#), Berry Schoenmakers and Andrey Sidorenko, IACR ePrint 2006/190.
- [Efficient Pseudorandom Generators Based on the DDH Assumption](#), Reza Rezaeian Farashahi and Berry Schoenmakers and Andrey Sidorenko, IACR ePrint 2006/321.
- [Analysis of the Linux Random Number Generator](#), Zvi Gutterman and Benny Pinkas and Tzachy Reinman.



The Wikibook *Cryptography* has a page on the topic of:
Random number generation



Cryptography portal

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: [Cryptographic algorithms](#) | [Cryptographically secure pseudorandom number generators](#)
| [Cryptographic primitives](#)

This page was last modified on 2 September 2015, at 01:11.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

