



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
[Български](#)
[Català](#)
[Čeština](#)
[Deutsch](#)
[Español](#)
[فارسی](#)
[Français](#)
[한국어](#)
[Hrvatski](#)
[Italiano](#)
[עברית](#)
[Nederlands](#)
[日本語](#)
[Norsk bokmål](#)
[Polski](#)
[Português](#)
[Русский](#)
[Simple English](#)
[Slovenščina](#)
[Српски / srpski](#)
[Suomi](#)
[Українська](#)
[中文](#)

[Edit links](#)

Article [Talk](#)

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

RC4

From Wikipedia, the free encyclopedia
(Redirected from [RC4 \(cipher\)](#))

This article is about the stream cipher. For other uses, see [RC4 \(disambiguation\)](#).

In [cryptography](#), **RC4** (Rivest Cipher 4 also known as **ARC4** or **ARCFOUR** meaning Alleged RC4, see below) is a [stream cipher](#). While remarkable for its simplicity and speed in software, multiple vulnerabilities have been discovered in RC4, rendering it insecure.^{[2][3]} It is especially vulnerable when the beginning of the output [keystream](#) is not discarded, or when nonrandom or related keys are used; some ways of using RC4 can lead to very insecure [protocols](#) such as [WEP](#).^[4]

As of 2015, there is speculation that some state cryptologic agencies may possess the capability to break RC4 even when used in the TLS protocol.^[5] [IETF](#) has published [RFC 7465](#) to prohibit the use of RC4 in TLS;^[2] [Mozilla](#) and [Microsoft](#) have issued similar recommendations.^{[6][7]}

In 2014, Ronald Rivest gave a talk and published a paper^[8] on an updated redesign called [Spritz](#).

RC4

General

Designers	Ron Rivest (RSA Security)
First published	Leaked in 1994 (designed in 1987)
Cipher detail	
Key sizes	40–2048 bits
State size	2064 bits (1684 effective)
Rounds	1
Speed	7 cycles per byte on original Pentium ^[1]

Contents

- 1 History
- 2 Description
 - 2.1 Key-scheduling algorithm (KSA)
 - 2.2 Pseudo-random generation algorithm (PRGA)
 - 2.3 RC4-based random number generators
 - 2.4 Implementation
 - 2.5 Test vectors
- 3 Security
 - 3.1 Roos' biases and key reconstruction from permutation
 - 3.2 Biased outputs of the RC4
 - 3.3 Fluhrer, Mantin and Shamir attack
 - 3.4 Klein's attack
 - 3.5 Combinatorial problem
 - 3.6 Royal Holloway attack
 - 3.7 Bar-mitzvah attack
 - 3.8 NOMORE attack
- 4 RC4 variants
 - 4.1 RC4A
 - 4.2 VMPC
 - 4.3 RC4⁺
 - 4.4 Spritz
- 5 RC4-based protocols
- 6 See also
- 7 References
- 8 Further reading
- 9 External links

History

RC4 was designed by [Ron Rivest](#) of [RSA Security](#) in 1987. While it is officially termed "Rivest Cipher 4", the RC acronym is alternatively understood to stand for "Ron's Code"^[9] (see also [RC2](#), [RC5](#) and [RC6](#)).

RC4 was initially a [trade secret](#), but in September 1994 a description of it was anonymously posted to the [Cypherpunks](#) mailing list.^[10] It was soon posted on the [sci.crypt newsgroup](#), and from there to many sites on the [Internet](#). The leaked code was confirmed to be genuine as its output was found to match that of proprietary

software using licensed RC4. Because the algorithm is known, it is no longer a [trade secret](#). The name *RC4* is [trademarked](#), so RC4 is often referred to as *ARCFOUR* or *ARC4* (meaning *alleged RC4*)^[11] to avoid trademark problems. [RSA Security](#) has never officially released the algorithm; Rivest has, however, linked to the [English Wikipedia](#) article on RC4 in his own course notes^[12] and confirmed the history of RC4 and its code in a paper by him.^[8] RC4 has become part of some commonly used encryption protocols and standards, including [WEP](#) and [WPA](#) for wireless cards and [TLS](#) (while it has been prohibited for all versions of TLS by [RFC 7465](#) [\[9\]](#), due to the [RC4 attacks](#) weakening or breaking RC4 used in SSL/TLS). The main factors in RC4's success over such a wide range of applications are its speed and simplicity: Efficient implementations in both software and hardware are very easy to develop.

Description [\[edit\]](#)

RC4 generates a [pseudorandom stream of bits](#) (a [keystream](#)). As with any stream cipher, these can be used for encryption by combining it with the plaintext using bit-wise [exclusive-or](#); decryption is performed the same way (since exclusive-or with given data is an [involution](#)). (This is similar to the [Vernam cipher](#) except that generated *pseudorandom bits*, rather than a prepared stream, are used.) To generate the keystream, the cipher makes use of a secret internal state which consists of two parts:

1. A [permutation](#) of all 256 possible [bytes](#) (denoted "S" below).
2. Two 8-bit index-pointers (denoted "i" and "j").

The permutation is initialized with a variable length [key](#), typically between 40 and 256 bits, using the [key-scheduling](#) algorithm (KSA). Once this has been completed, the stream of bits is generated using the *pseudo-random generation algorithm* (PRGA).

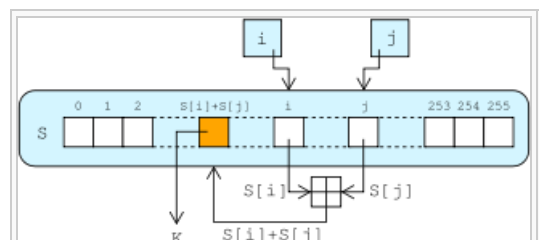
Key-scheduling algorithm (KSA) [\[edit\]](#)

The [key-scheduling](#) algorithm is used to initialize the permutation in the array "S". "keylength" is defined as the number of bytes in the key and can be in the range $1 \leq \text{keylength} \leq 256$, typically between 5 and 16, corresponding to a [key length](#) of 40 – 128 bits. First, the array "S" is initialized to the [identity permutation](#). S is then processed for 256 iterations in a similar way to the main PRGA, but also mixes in bytes of the key at the same time.

```
for i from 0 to 255
  S[i] := i
endfor
j := 0
for i from 0 to 255
  j := (j + S[i] + key[i mod keylength]) mod 256
  swap values of S[i] and S[j]
endfor
```

Pseudo-random generation algorithm (PRGA) [\[edit\]](#)

For as many iterations as are needed, the PRGA modifies the state and outputs a byte of the keystream. In each iteration, the PRGA increments *i*, looks up the *i*th element of S, *S*[*i*], and adds that to *j*, exchanges the values of *S*[*i*] and *S*[*j*], and then uses the sum *S*[*i*] + *S*[*j*] (modulo 256) as an index to fetch a third element of S, (the keystream value *K* below) which is XORed with the next byte of the message to produce the next byte of either ciphertext or plaintext. Each element of S is swapped with another element at least once every 256 iterations.



The lookup stage of RC4. The output byte is selected by looking up the values of *S*(*i*) and *S*(*j*), adding them together modulo 256, and then using the sum as an index into S; *S*(*S*(*i*) + *S*(*j*)) is used as a byte of the keystream, *K*.

```
i := 0
j := 0
while GeneratingOutput:
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap values of S[i] and S[j]
  K := S[(S[i] + S[j]) mod 256]
  output K
```

RC4-based random number generators [\[edit\]](#)

Several [operating systems](#) include arc4random, an API originating in [OpenBSD](#) providing access to a random number generator originally based on RC4. In OpenBSD 5.5, released in [May 2014](#), arc4random was modified to use [ChaCha20](#).^{[13][14]} As of January 2015, implementation of arc4random in [NetBSD](#)^{[15][16]} also uses ChaCha20, however, implementation of arc4random in [FreeBSD](#),^[17] [Linux](#)'s libbsd,^[18] and [Mac OS X](#)^[19] are still based on RC4.

Proposed new random number generators are often compared to the RC4 random number generator.^{[20][21]}

Unfortunately, several attacks on RC4 are able to [distinguish its output from a random sequence](#).^[22]

Implementation [\[edit\]](#)

Many stream ciphers are based on [linear feedback shift registers](#) (LFSRs), which, while efficient in hardware, are less so in software. The design of RC4 avoids the use of LFSRs, and is ideal for software implementation, as it requires only byte manipulations. It uses 256 bytes of memory for the state array, S[0] through S[255], k bytes of memory for the key, key[0] through key[k-1], and integer variables, i, j, and K. Performing a modular reduction of some value modulo 256 can be done with a [bitwise AND](#) with 255 (which is equivalent to taking the low-order byte of the value in question).

Test vectors [\[edit\]](#)

These test vectors are not official, but convenient for anyone testing their own RC4 program. The keys and plaintext are [ASCII](#), the keystream and ciphertext are in [hexadecimal](#).

Key	Keystream	Plaintext	Ciphertext
Key	EB9F7781B734CA72A719...	Plaintext	BBF316E8D940AF0AD3
Wiki	6044DB6D41B7...	pedia	1021BF0420
Secret	04D46B053CA87B59...	Attack at dawn	45A01F645FC35B383552544B9BF5

Security [\[edit\]](#)

Unlike a modern stream cipher (such as those in [eSTREAM](#)), RC4 does not take a separate [nonce](#) alongside the key. This means that if a single long-term key is to be used to securely encrypt multiple streams, the protocol must specify how to combine the nonce and the long-term key to generate the stream key for RC4. One approach to addressing this is to generate a "fresh" RC4 key by [hashing](#) a long-term key with a [nonce](#). However, many applications that use RC4 simply concatenate key and nonce; RC4's weak [key schedule](#) then gives rise to [related key attacks](#), like the [Fluhrer, Mantin and Shamir attack](#) (which is famous for breaking the [WEP](#) standard).^[23]

Because RC4 is a [stream cipher](#), it is more [malleable](#) than common [block ciphers](#). If not used together with a strong [message authentication code](#) (MAC), then encryption is vulnerable to a [bit-flipping attack](#). The cipher is also vulnerable to a [stream cipher attack](#) if not implemented correctly.^[24] Furthermore, inadvertent double encryption of a message with the same key may accidentally output plaintext rather than ciphertext because the [involutory](#) nature of the XOR function would result in the second operation reversing the first.

It is noteworthy, however, that RC4, being a stream cipher, was for a period of time the only common cipher that was immune^[25] to the 2011 [BEAST attack](#) on [TLS 1.0](#). The attack exploits a known weakness in the way [cipher block chaining mode](#) is used with all of the other ciphers supported by TLS 1.0, which are all block ciphers.

In March 2013, there were new attack scenarios proposed by Isobe, Ohigashi, Watanabe and Morii,^[26] as well as AlFardan, Bernstein, Paterson, Poettering and Schuldts that use new statistical biases in RC4 key table^[27] to recover plaintext with large number of TLS encryptions.^{[28][29]}

The use of RC4 in TLS is prohibited by [RFC 7465](#) published in February 2015.

Roos' biases and key reconstruction from permutation [\[edit\]](#)

In 1995, Andrew Roos experimentally observed that the first byte of the keystream is correlated to the first three bytes of the key and the first few bytes of the permutation after the KSA are correlated to some linear combination of the key bytes.^[30] These biases remained unproven until 2007, when Goutam Paul, Siddheshwar Rath and Subhamoy Maitra^[31] proved the keystream-key correlation and in another work Goutam Paul and Subhamoy Maitra^[32] proved the permutation-key correlations. The latter work also used the permutation-key correlations to design the first algorithm for complete key reconstruction from the final permutation after the KSA, without any assumption on the key or IV. This algorithm has a constant probability of success in a time which is the square root of the exhaustive key search complexity. Subsequently, many other works have been performed on key reconstruction from RC4 internal states.^{[33][34][35]} Subhamoy Maitra and Goutam Paul^[36] also showed that the Roos type biases still persist even when one considers nested permutation indices, like $S[S[i]]$ or $S[S[S[i]]]$. These types of biases are used in some of the later key reconstruction methods for increasing the success probability.

Biased outputs of the RC4 [\[edit\]](#)

The keystream generated by the RC4 is biased in varying degrees towards certain sequences making it vulnerable to [distinguishing attacks](#). The best such attack is due to Itsik Mantin and [Adi Shamir](#) who showed that the second output byte of the cipher was biased toward zero with probability 1/128 (instead of 1/256). This is due to the fact that if the third byte of the original state is zero, and the second byte is not equal to 2, then the second output byte is always zero. Such bias can be detected by observing only 256 bytes.^[22]

[Souradyuti Paul](#) and [Bart Preneel](#) of COSIC showed that the first and the second bytes of the RC4 were also biased. The number of required samples to detect this bias is 2^{25} bytes.^[37]

[Scott Fluhrer](#) and David McGrew also showed such attacks which distinguished the keystream of the RC4 from a random stream given a gigabyte of output.^[38]

The complete characterization of a single step of RC4 PRGA was performed by Riddhipratim Basu, Shirshendu Ganguly, Subhamoy Maitra, and Goutam Paul.^[39] Considering all the permutations, they prove that the distribution of the output is not uniform given i and j , and as a consequence, information about j is always leaked into the output.

Fluhrer, Mantin and Shamir attack [\[edit\]](#)

Main article: [Fluhrer, Mantin and Shamir attack](#)

In 2001, a new and surprising discovery was made by [Fluhrer](#), [Mantin](#) and [Shamir](#): over all possible RC4 keys, the statistics for the first few bytes of output keystream are strongly non-random, leaking information about the key. If the nonce and long-term key are simply concatenated to generate the RC4 key, this long-term key can be discovered by analysing a large number of messages encrypted with this key.^[40] This and related effects were then used to break the WEP ("wired equivalent privacy") encryption used with [802.11 wireless networks](#). This caused a scramble for a standards-based replacement for WEP in the 802.11 market, and led to the [IEEE 802.11i](#) effort and [WPA](#).^[41]

Protocols can defend against this attack by discarding the initial portion of the keystream. Such a modified algorithm is traditionally called "RC4-drop[n]", where n is the number of initial keystream bytes that are dropped. The SCAN default is $n = 768$ bytes, but a conservative value would be $n = 3072$ bytes.^[42]

The Fluhrer, Mantin and Shamir attack does not apply to RC4-based SSL, since SSL generates the encryption keys it uses for RC4 by hashing, meaning that different SSL sessions have unrelated keys.^[43]

Klein's attack [\[edit\]](#)

In 2005, Andreas Klein presented an analysis of the RC4 stream cipher showing more correlations between the RC4 keystream and the key.^[44] [Erik Tews](#), [Ralf-Philipp Weinmann](#), and [Andrei Pyshkine](#) used this analysis to create aircrack-ptw, a tool which cracks 104-bit RC4 used in 128-bit WEP in under a minute.^[45] Whereas the Fluhrer, Mantin, and Shamir attack used around 10 million messages, aircrack-ptw can break 104-bit keys in 40,000 frames with 50% probability, or in 85,000 frames with 95% probability.

Combinatorial problem [\[edit\]](#)

A combinatorial problem related to the number of inputs and outputs of the RC4 cipher was first posed by [Itsik Mantin](#) and [Adi Shamir](#) in 2001, whereby, of the total 256 elements in the typical state of RC4, if x number of

elements ($x \leq 256$) are *only* known (all other elements can be assumed empty), then the maximum number of elements that can be produced deterministically is also x in the next 256 rounds. This conjecture was put to rest in 2004 with a formal proof given by [Souradyuti Paul](#) and [Bart Preneel](#).^[46]

Royal Holloway attack [\[edit\]](#)

In 2013, a group of security researchers at the Information Security Group at Royal Holloway, University of London reported an attack that can become effective using only 2^{24} connections.^{[47][48][49]} While yet not a practical attack for most purposes, this result is sufficiently close to one that it has led to speculation that it is plausible that some state cryptologic agencies may already have better attacks that render RC4 insecure.^[5] Given that as of 2013 a large amount of [TLS](#) traffic uses RC4 to avoid recent attacks on block ciphers that use [cipher block chaining](#), if these hypothetical better attacks exist, then this would make the TLS-with-RC4 combination insecure against such attackers in a large number of practical scenarios.^[5]

In March 2015 researcher to Royal Holloway announced improvements to their attack, providing a 2^{26} attack against passwords encrypted with RC4, as used in TLS.^[50]

Bar-mitzvah attack [\[edit\]](#)

Main article: [Bar-mitzvah attack](#)

On the Black Hat Asia 2015, Itsik Mantin presented another attack against SSL using RC4 cipher.^{[51][52]}

NOMORE attack [\[edit\]](#)

In 2015, security researchers from [KU Leuven](#) presented new attacks against RC4 in both [TLS](#) and [WPA-TKIP](#).^[53] Dubbed the Numerous Occurrence MONitoring & Recovery Exploit (NOMORE) attack, it is the first attack of its kind that was demonstrated in practice. Their attack against [TLS](#) can decrypt a secure [HTTP cookie](#) within 75 hours. The attack against WPA-TKIP can be completed within an hour, and allows an attacker to decrypt and inject arbitrary packets.

RC4 variants [\[edit\]](#)

As mentioned above, the most important weakness of RC4 comes from the insufficient key schedule; the first bytes of output reveal information about the key. This can be corrected by simply discarding some initial portion of the output stream.^[54] This is known as RC4-drop N , where N is typically a multiple of 256, such as 768 or 1024.

A number of attempts have been made to strengthen RC4, notably Spritz, RC4A, [VMPC](#), and RC4+.

RC4A [\[edit\]](#)

[Souradyuti Paul](#) and [Bart Preneel](#) have proposed an RC4 variant, which they call RC4A.^[55]

RC4A uses two state arrays $S1$ and $S2$, and two indexes $j1$ and $j2$. Each time i is incremented, two bytes are generated:

1. First, the basic RC4 algorithm is performed using $S1$ and $j1$, but in the last step, $S1[j] + S1[j1]$ is looked up in $S2$.
2. Second, the operation is repeated (without incrementing i again) on $S2$ and $j2$, and $S1[S2[j]+S2[j2]]$ is output.

Thus, the algorithm is:

```
All arithmetic is performed modulo 256
i := 0
j1 := 0
j2 := 0
while GeneratingOutput:
    i := i + 1
    j1 := j1 + S1[i]
    swap values of S1[i] and S1[j1]
    output S2[S1[i] + S1[j1]]
    j2 := j2 + S2[i]
    swap values of S2[i] and S2[j2]
    output S1[S2[i] + S2[j2]]
endwhile
```

Although the algorithm required the same number of operations per output byte, there is greater parallelism

than RC4, providing a possible speed improvement.

Although stronger than RC4, this algorithm has also been attacked,^[56] with Alexander Maximov^[57] and a team from NEC^[58] developing ways to distinguish its output from a truly random sequence.

VMPC [\[edit\]](#)

"[Variably Modified Permutation Composition](#)" is another RC4 variant.^[59] It uses similar key schedule as RC4, with $j := S[(j + S[j] + \text{key}[i \bmod \text{keylength}]) \bmod 256]$ iterating $3 \times 256 = 768$ times rather than 256, and with an optional additional 768 iterations to incorporate an initial vector. The output generation function operates as follows:

```
All arithmetic is performed modulo 256.
i := 0
while GeneratingOutput:
    a := S[i]
    j := S[j + a]
    b := S[j]
    output S[S[b] + 1]
    S[i] := b      (Swap S[i] and S[j])
    S[j] := a
    i := i + 1
endwhile
```

This was attacked in the same papers as RC4A, and can be distinguished within 2^{38} output bytes.^{[56][58]}

RC4⁺ [\[edit\]](#)

RC4⁺ is a modified version of RC4 with a more complex three-phase key schedule (taking about 3× as long as RC4, or the same as RC4-drop512), and a more complex output function which performs four additional lookups in the S array for each byte output, taking approximately 1.7× as long as basic RC4.^[60]

```
All arithmetic modulo 256. << and >> are left and right shift, @ is exclusive OR
while GeneratingOutput:
    i := i + 1
    a := S[i]
    j := j + a
    b := S[j]
    S[i] := b      (Swap S[i] and S[j])
    S[j] := a
    c := S[i<<5 @ j>>3] + S[j<<5 @ i>>3]
    output (S[a+b] + S[c@0xAA) @ S[j+b]
endwhile
```

This algorithm has not been analyzed significantly.

Spritz [\[edit\]](#)

Ron Rivest and Jacob Schuldt have proposed replacing RC4 with an improved and slightly modified version:^[8]

```
All arithmetic is performed modulo 256
while GeneratingOutput:
    i := i + w
    j := k + S[j + S[i]]
    k := k + i + S[j]
    swap values of S[i] and S[j]
    output z := S[j + S[i + S[z + k]]]
endwhile
```

```
Pseudocode with modulo 256 arithmetic
while GeneratingOutput:
    i := (i + w) mod 256
    j := (k + S[(j + S[i]) mod 256]) mod 256
    k := (k + i + S[j]) mod 256
    swap values of S[i] and S[j]
    output z := S[(j + S[(i + S[(z + k) mod 256]) mod 256]) mod 256]
```


endwhile

The value w , is relatively prime to the size of the S array. So after 256 iterations of this inner loop, the value i (incremented by w every iteration) has taken on all possible values $0..255$, and every byte in the S array has been swapped at least once.

Like other [sponge functions](#), Spritz can be used to build a cryptographic hash function, a deterministic random bit generator ([DRBG](#)), an encryption algorithm that supports [authenticated encryption](#) with associated data (AEAD), etc.^[8]

RC4-based protocols [\[edit\]](#)

- [WEP](#)
- [WPA](#) (default algorithm, but can be configured to use [AES-CCMP](#) instead of RC4)
- [BitTorrent protocol encryption](#)
- [Microsoft Office XP](#) (insecure implementation since nonce remains unchanged when documents get modified^[61])
- [Microsoft Point-to-Point Encryption](#)
- [Transport Layer Security / Secure Sockets Layer](#) (was optional and then the use of RC4 was prohibited in [RFC 7465](#) [↗](#))
- [Secure Shell](#) (optionally)
- [Remote Desktop Protocol](#)
- [Kerberos](#) (optionally)
- [SASL Mechanism Digest-MD5](#) (optionally, *historic*, obsoleted in [RFC 6331](#) [↗](#))
- [Gpcode.AK](#), an early June 2008 computer virus for Microsoft Windows, which takes documents hostage for [ransom](#) by obscuring them with RC4 and RSA-1024 encryption
- [PDF](#)
- [Skype](#) (in modified form)^[62]

Where a protocol is marked with "(optionally)", RC4 is one of multiple ciphers the system can be configured to use.

See also [\[edit\]](#)



















- [eSTREAM](#) - An evaluation of new [stream ciphers](#) being conducted by the EU.
- [TEA](#), [Block TEA](#) also known as [eXtended TEA](#) and [Corrected Block TEA](#) - A family of [block ciphers](#) that, like RC4, are designed to be very simple to implement.
- [Advanced Encryption Standard](#)
- [CipherSaber](#)
- [Spritz](#) a spongy RC4-like [stream cipher](#) and [hash function](#) by Ronald L. Rivest and Jacob C. N. Schuldt.

References [\[edit\]](#)

- ↑ P. Prasithsangaree and P. Krishnamurthy (2003). "Analysis of Energy Consumption of RC4 and AES Algorithms in Wireless LANs" [↗](#) (PDF). Archived from the original [↗](#) (PDF) on 2013-12-03.
- ↑ *a* *b* Andrei Popov (February 2015). *Prohibiting RC4 Cipher Suites* [↗](#). RFC 7465.
- ↑ Lucian Constantin (14 May 2014). "Microsoft continues RC4 encryption phase-out plan with .NET security updates" [↗](#). *ComputerWorld*.
- ↑ J. Katz, Y. Lindell (2014), *Introduction to Modern Cryptography*, Chapman and Hall/CRC, p. 77
- ↑ *a* *b* *c* John Leyden (2013-09-06). "That earth-shattering NSA crypto-cracking: Have spooks smashed RC4?" [↗](#). The Register.
- ↑ "Mozilla Security Server Side TLS Recommended Configurations" [↗](#). Mozilla. Retrieved 2015-01-03.
- ↑ "Security Advisory 2868725: Recommendation to disable RC4" [↗](#). Microsoft. 2013-11-12. Retrieved 2013-12-04.
- ↑ *a* *b* *c* *d* Rivest, Ron; Schuldt, Jacob (2014-10-27). "Spritz - a spongy RC4-like stream cipher and hash function" [↗](#) (PDF). Retrieved 26 October 2014.
- ↑ [Rivest FAQ](#) [↗](#)
- ↑ "Thank you Bob Anderson" [↗](#). *Cypherpunks* (Mailing list). 1994-09-09. Retrieved 2007-05-28.
- ↑ "Manual Pages: arc4random" [↗](#). 2013-06-05. Retrieved 23 June 2014.
- ↑ 6.857 Computer and Network Security Spring 2008: Lectures and Handouts [↗](#)
- ↑ "OpenBSD 5.5" [↗](#). Retrieved 21 September 2014.
- ↑ [deraadt](#), ed. (2014-07-21). "libc/crypt/arc4random.c" [↗](#). *BSD Cross Reference*, *OpenBSD src/lib/*. Retrieved 2015-01-13. "ChaCha based random number generator for OpenBSD."
- ↑ [riastradh](#). ed. (2014-11-16). "libc/aen/arc4random.c" [↗](#). *BSD Cross Reference*. *NetBSD src/lib/*. Retrieved

- 2015-01-13. "Legacy arc4random(3) API from OpenBSD reimplemented using the ChaCha20 PRF, with per-thread state."
16. [^] ["arc4random - NetBSD Manual Pages"](#) . Retrieved 6 January 2015.
17. [^] ["arc4random"](#) . Retrieved 6 January 2015.
18. [^] ["arc4random\(3\): arc4 random number generator - Linux man page"](#) . Retrieved 6 January 2015.
19. [^] ["arc4random\(3\) Mac OS X Developer Tools Manual Page"](#) . Retrieved 6 January 2015.
20. [^] Bartosz Zoltak. "VMPC-R: Cryptographically Secure Pseudo-Random Number Generator, Alternative to RC4". 2010?
21. [^] Chefranov, A.G. "Pseudo-Random Number Generator RC4 Period Improvement" . 2006.
22. [^] ^a ^b Itsik Martin, Adi Shamir (2001). "A Practical Attack on Broadcast RC4" (PDF). pp. 152 – 164.
23. [^] "RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4" . RSA Laboratories. 1 September 2001.
24. [^] Sklyarov, Dmitry (2004). *Hidden Keys to Software Break-Ins and Unauthorized Entry* . A-List Publishing. pp. 92–93. ISBN 1931769303.
25. [^] <http://serverfault.com/questions/315042/>
26. [^] Isobe, Takanori; Ohigashi, Toshihiro (Mar 10–13, 2013). "Security of RC4 Stream Cipher" . Hiroshima University. Retrieved 2014-10-27.
27. [^] Pouyan Sepehrdad, Serge Vaudenay, Martin Vuagnoux (2011). "Discovery and Exploitation of New Biases in RC4" . *Lecture Notes in Computer Science* **6544**: 74–91. doi:10.1007/978-3-642-19574-7_5 .
28. [^] Green, Matthew. "Attack of the week: RC4 is kind of broken in TLS" . *Cryptography Engineering*. Retrieved March 12, 2013.
29. [^] Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering and Jacob Schuld. "On the Security of RC4 in TLS" . Royal Holloway University of London. Retrieved March 13, 2013.
30. [^] Andrew Roos. A Class of Weak Keys in the RC4 Stream Cipher. Two posts in sci.crypt, message-id 43u1eh\$1j3@hermes.is.co.za and 44ebge\$1lf@hermes.is.co.za, 1995.
31. [^] Goutam Paul, Siddheshwar Rathi and Subhamoy Maitra. On Non-negligible Bias of the First Output Byte of RC4 towards the First Three Bytes of the Secret Key. Proceedings of the International Workshop on Coding and Cryptography (WCC) 2007, pages 285-294 and Designs, Codes and Cryptography Journal, pages 123-134, vol. 49, no. 1-3, December 2008.
32. [^] Goutam Paul and Subhamoy Maitra. Permutation after RC4 Key Scheduling Reveals the Secret Key. SAC 2007, pages 360-377, vol. 4876, *Lecture Notes in Computer Science*, Springer.
33. [^] Eli Biham and Yaniv Carmeli. Efficient Reconstruction of RC4 Keys from Internal States. FSE 2008, pages 270-288, vol. 5086, *Lecture Notes in Computer Science*, Springer.
34. [^] Mete Akgun, Pinar Kavak, Huseyin Demirci. New Results on the Key Scheduling Algorithm of RC4. INDOCRYPT 2008, pages 40-52, vol. 5365, *Lecture Notes in Computer Science*, Springer.
35. [^] Riddhipratim Basu, Subhamoy Maitra, Goutam Paul and Tanmoy Talukdar. On Some Sequences of the Secret Pseudo-random Index j in RC4 Key Scheduling. Proceedings of the 18th International Symposium on Applied Algebra, Algebraic Algorithms and Error Correcting Codes (AAECC), June 8–12, 2009, Tarragona, Spain, pages 137-148, vol. 5527, *Lecture Notes in Computer Science*, Springer.
36. [^] Subhamoy Maitra and Goutam Paul. New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. Proceedings of the 15th Fast Software Encryption (FSE) Workshop, February 10–13, 2008, Lausanne, Switzerland, pages 253-269, vol. 5086, *Lecture Notes in Computer Science*, Springer.
37. [^] Souradyuti Paul, Bart Preneel. "Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator" (PDF). pp. 52 – 67.
38. [^] Scott R. Fluhrer, David A. McGrew. "Statistical Analysis of the Alleged RC4 Keystream Generator" (PDF). pp. 19 – 30.
39. [^] Basu, Riddhipratim; Ganguly, Shirshendu; Maitra, Subhamoy; Paul, Goutam (2008). "A Complete Characterization of the Evolution of RC4 Pseudo Random Generation Algorithm". *Journal of Mathematical Cryptology* **2** (3): 257–289. doi:10.1515/JMC.2008.012 .
40. [^] Scott R. Fluhrer, Itsik Martin and Adi Shamir, Weaknesses in the Key Scheduling Algorithm of RC4. *Selected Areas in Cryptography* 2001, pp1 – 24 (PS) .
41. [^] Interim technology for wireless LAN security: WPA to replace WEP while industry develops new security standard
42. [^] "RC4-drop(nbytes)" in the "Standard Cryptographic Algorithm Naming" database
43. [^] Ron Rivest. *RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4* .
44. [^] A. Klein, Attacks on the RC4 stream cipher, *Designs, Codes and Cryptography* (2008) 48:269-286
45. [^] Erik Tews, Ralf-Philipp Weinmann, Andrei Pyshkin. *Breaking 104-bit WEP in under a minute* .
46. [^] Souradyuti Paul and Bart Preneel, A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. *Fast Software Encryption - FSE 2004*, pp245 – 259 (PDF) .
47. [^] John Leyden (2013-03-15). "HTTPS cookie crypto CRUMBLES AGAIN in hands of stats boffins" . The Register.
48. [^] AlFardan et al. (2013-07-08). "On the Security of RC4 in TLS and WPA" (PDF). Information Security Group, Royal Holloway, University of London.
49. [^] "On the Security of RC4 in TLS and WPA" . Information Security Group, Royal Holloway, University of London. Retrieved 2013-09-06. (archived)

retrieved 2013-09-06. (website)

50. [^] <http://www.isg.rhul.ac.uk/tls/RC4mustdie.html> 
51. [^] <https://www.blackhat.com/asia-15/briefings.html#bar-mitzva-attack-breaking-ssl-with-13-year-old-rc4-weakness> 
52. [^] https://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf 
53. [^] Mathy Vanhoef and Frank Piessens (2015-08-09). "RC4 NOMORE: Numerous Occurrence MOnitoring & Recovery Exploit" 
54. [^] Ilya Mironov (2002-06-01), "(Not So) Random Shuffles of RC4" , *Advances in Cryptology – CRYPTO 2002*, Lecture Notes in Computer Science **2442**, Springer-Verlag, pp. 304–319, doi:10.1007/3-540-45708-9_20 , ISBN 3-540-44050-X, Cryptology ePrint Archive: Report 2002/067, retrieved 2011-11-04
55. [^] Souradyuti Paul; Bart Preneel (2004), "A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher" , *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science **3017**, Springer-Verlag, pp. 245–259, doi:10.1007/978-3-540-25937-4_16 , ISBN 3-540-22171-9, retrieved 2011-11-04
56. ^{^ a b} [CryptoLounge: RC4A](#) 
57. [^] Alexander Maximov (2007-02-22), *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers* , Cryptology ePrint Archive: Report 2007/070, retrieved 2011-11-04
58. ^{^ a b} Yukiyasu Tsunoo; Teruo Saito; Hiroyasu Kubo; Maki Shigeri; Tomoyasu Suzaki; Takeshi Kawabata (2005), *The Most Efficient Distinguishing Attack on VMPC and RC4A*  (PDF)
59. [^] Bartosz Zoltak (2004), "VMPC One-Way Function and Stream Cipher"  (PDF), *Fast Software Encryption, FSE 2004*, Lecture Notes in Computer Science **3017**, Springer-Verlag, pp. 210–225, doi:10.1007/978-3-540-25937-4_14 , ISBN 3-540-22171-9, retrieved 2011-11-04
60. [^] Subhamoy Maitra; Goutam Paul (2008-09-19), "Analysis of RC4 and Proposal of Additional Layers for Better Security Margin" , *Progress in Cryptology – INDOCRYPT 2008*, Lecture Notes in Computer Science **5365**, Springer-Verlag, pp. 27–39, doi:10.1007/978-3-540-89754-5_3 , ISBN 3-540-89753-4, Cryptology ePrint Archive: Report 2008/396, retrieved 2011-11-04
61. [^] Hongjun Wu, "The Misuse of RC4 in Microsoft Word and Excel". <http://eprint.iacr.org/2005/007> 
62. [^] "Skype's encryption procedure partly exposed" . www.h-online.com. Archived from the original  on 11 July 2010. Retrieved 2010-07-08.

Further reading [[edit](#)]

- Paul, Goutam; Subhamoy Maitra (2011). *RC4 Stream Cipher and Its Variants*[↗](#). CRC Press. ISBN 9781439831359.
- Schneier, Bruce (1995). "Chapter 17 - Other Stream Ciphers and Real Random-Sequence Generators". *Applied Cryptography: Protocols, Algorithms, and Source Code in C*[↗](#) (2nd ed.). Wiley. ISBN 978-0471117094.

External links [[edit](#)]

- Original posting of RC4 algorithm to Cypherpunks mailing list[↗](#), Archived version[↗](#)
- RFC 4345[↗](#) - Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol
- RFC 6229[↗](#) - Test Vectors for the Stream Cipher RC4
- RFC 7465[↗](#) - Prohibiting RC4 Cipher Suites
- IETF Draft - A Stream Cipher Encryption Algorithm "Arcfour"[↗](#)
- SCAN's entry for RC4[↗](#)
- Attacks on RC4[↗](#)
- RC4 - Cryptology Pointers by Helger Lipmaa[↗](#)
- RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4[↗](#)

RC4 in WEP

- (in)Security of the WEP algorithm[↗](#)
- Fluhrer, Mantin, and Shamir attack on WEP (postscript format)[↗](#)

v · t · e	Stream ciphers
Widely used ciphers	RC4 · Block ciphers in stream mode
eSTREAM Portfolio	Software HC-256 · Rabbit · Salsa20 · SOSEMANUK
	Hardware Grain · MCKEY · Trivium
Other ciphers	A5/1 · A5/2 · Achterbahn · E0 · F-FCSR · FISH · ISAAC · MJOL · Panama · Phelix · Pike · Py · QUAD · Scream · SEAL · SNOW · SOBER · SOBER-128 · VEST · WAKE
Theory	Shift register · LFSR · NLFSR · Shrinking generator · T-function · IV
Attacks	Correlation attack · Correlation immunity
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: [Stream ciphers](#) | [Broken stream ciphers](#) | [Pseudorandom number generators](#) | [Free ciphers](#)

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](#)

