**Key-scheduling algorithm (KSA)**[[edit](https://en.wikipedia.org/w/index.php?title=RC4&action=edit&section=3)]

The [key-scheduling](https://en.wikipedia.org/wiki/Key_schedule) 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 ≤ keylength ≤ 256, typically between 5 and 16, corresponding to a [key length](https://en.wikipedia.org/wiki/Key_length) of 40 – 128 bits. First, the array "S" is initialized to the [identity permutation](https://en.wikipedia.org/wiki/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

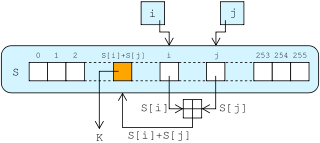
**for** i **from** 0 **to** 255

j := (j + S[i] + key[i [mod](https://en.wikipedia.org/wiki/Modulo_operation) keylength]) mod 256

swap values of S[i] and S[j]

**endfor**

**Pseudo-random generation algorithm (PRGA)**[[edit](https://en.wikipedia.org/w/index.php?title=RC4&action=edit&section=4)]

[](https://en.wikipedia.org/wiki/File:RC4.svg)

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 key stream, K.

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*] 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)
* then bitwise exclusive ORed ([XORed](https://en.wikipedia.org/wiki/Exclusive_or" \o "Exclusive or)) 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.

i := 0

j := 0

**while** GeneratingOutput:

i := (i + 1) mod 256

j := (j + S[i]) mod 256

[swap values](https://en.wikipedia.org/wiki/Swap_(computer_science)) of S[i] and S[j]

K := S[(S[i] + S[j]) mod 256]

output K

**endwhile**

Thus, this produces a stream of K[0],K[1],... which are [XOR](https://en.wikipedia.org/wiki/Exclusive_or" \o "Exclusive or)'ed with the *plaintext* to obtain the *ciphertext*. So ciphertext[*l*] = plaintext[*l*] ⊕ K[*l*] .