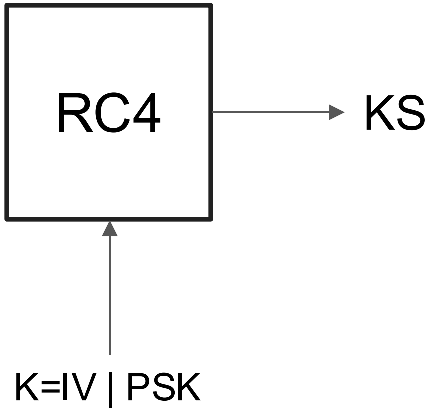
WEP Attack against RC4
Ethical Hacking 24/25

Prof. Daniele Friolo
Sapienza University of Rome
Cybersecurity Course

Introducing RC4

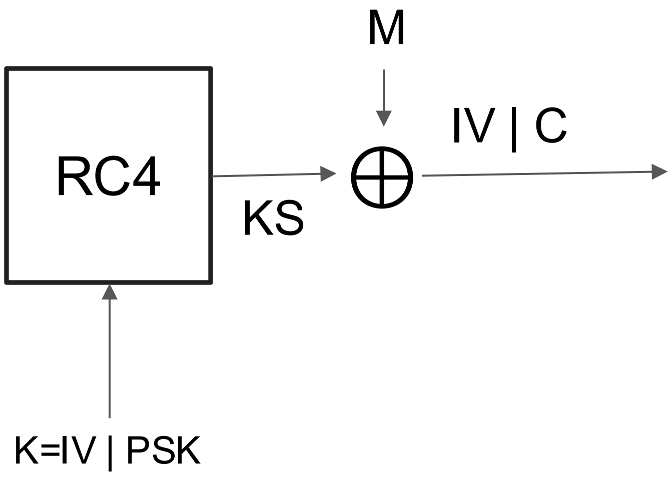
- Designed by Ron Rivest in 1987
- Symmetric stream cipher
- Initally kept secret and never released publicly
- Description of the cipher posted anonymously in 1994 and later confirmed to be compatible
- Simple to implement, fast
- Today RC4 should not be used anymore due to the various nature of attacks

Introducing RC4



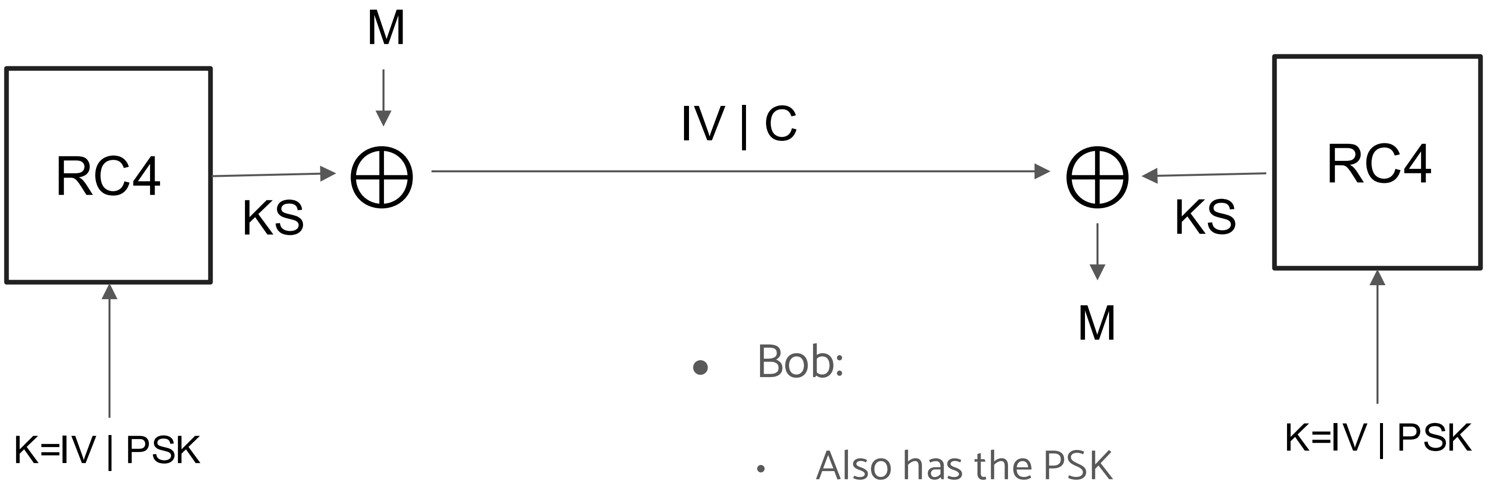
- WEP
  - Uses RC4 with 128bit key (16 bytes)
  - PSK = WEP Pre-Shared Key
  - IV is 3 bytes

Introducing RC4



- Alice:
  - Generates a keystream $KS = RC4(IV \mid M)$
  - Encrypts $C = M \oplus KS$
  - Sends IV, C to Bob (with an integrity check)

Introducing RC4



- Bob:
  - Also has the PSK
  - Generate $KS = RC4(K)$
  - Decrypts $M = KS \oplus C$

RC4 Algorithm

- Divided in two phases:

  - S-Box generation algorithm (Key State Algorithm) from the IV and the PSK
    Idea: the IV, together with the PSK is used to produce a unique and pseudorandom-random scrambled S-Box (a pseud–random state vector)

  - Pseudo-Random Number Generator (PRNGA) from S-Box to produce KS
    Idea: Using the S-Box as a seed, produce an arbitrary Keystream KS to be XORed with the message M

- New KS for new message from the same S-Box

- In WEP, S-Boxes are freshly generated from new IVs for each packet

RC4 Algorithm

Key Scheduling Algorithm

- $S_i[k]$: k-th value of S after running KSA for i steps
- i and j are modulo 256, i in K[i] is mod keylength (typically 5-16 bytes)
- Recall: K = IV | PSK where IV is 24 bits (3B)

for i = 0 to 255 **do**
$S_o[i] = i$
$j_0 = 0$
for i = 0 to 255 **do**
$j_{i+1} = (j_i + S_i[i] + K[i])$
$S_{i+1}[i] = S_i[j_{i+1}]$
$S_{i+1}[j_{i+1}] = S_i[i]$

Swap $S[i], S[j]$

S_0	0	1	2	3	4	….	255		
S_1	180	1	2	3	4	….	0	…	255
S_2	180	4	2	3	1	….	0	…	255

$K[0] = 180$ $\quad j_0 = 0 \quad j_1 = 0 + 0 + 180 = 180$

$K[1] = 80$ $\quad j_1 = 180 \quad j_2 = 180 + 80\ (mod256) = 4$

RC4 Algorithm

Pseudo-Random Generation Algorithm

$i = 0$
$j = 0$
while *output* **do**
  $i = (i + 1)\ mod256$
  $j = (j + S[i])\ mod256$
  $\text{swap}(S[i], S[j])$
  $\text{output}\ S[(S[i] + S[j])\ mod256]$

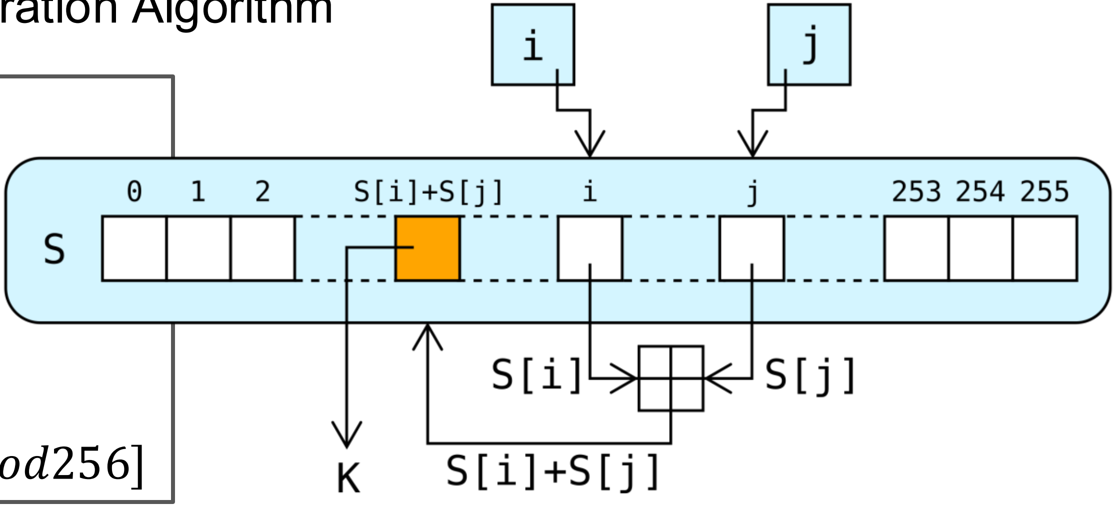


Image source: Wikipedia

Weaknesses in the KSA of RC4 [FMS01]

- Attack from Fluhrer, Mantin and Shamir (FMS)
- RC4 can leak key information from the keystream
- At the time of publications, vendors said it was “impractical”
- Real-world implementation appeared soon thereafter
- Today, full PSK recovery from WEP
- Need for a lot of IV to recover. However, ARP Spoofing + replay attacks can induce a host to send many packets!

Weaknesses in the KSA of RC: Idea

- Since ciphertexts contain the IV, that is part of the key K= IV I PSK, any snooping adversary have access to the first three bytes used by the KSA

- Certain weak IVs are known to leak some info about the PSK

- Enough packets → Full recovery of the PSK

- Observe that we can run KSA for 3 steps, since we know the first **3 bytes** of K (the IV).

- Assume that the KSA is run for L = 3 steps. *What happens in the L+1-th step*?

for i = 0 to 255 **do**

$$j_{i+1} = (j_i + S_i[i] + K[i])$$
$$S_{i+1}[i] = S_i[j_{i+1}]$$
$$S_{i+1}[j_{i+1}] = S_i[i]$$

Partial Execution of the KSA

for i = 0 to 255 **do**

$S_o[i] = i$

$j_0 = 0$

for i = 0 to 255 **do**

$j_{i+1} = (j_i + S_i[i] + K[i])$

$S_{i+1}[i] = S_i[j_{i+1}]$

$S_{i+1}[j_{i+1}] = S_i[i]$

$$i = L$$

$$j_{L+1} = j_L + S_L[i] + K[i]$$
$$= j_L + S_L[L] + K[L]$$

So

$$K[L] = j_{L+1} - j_L - S_L[L]$$

- j_L and $S_L[L]$ known
- If we knew j_{L+1} we could obtain $K[L]$, i.e. the next byte of the key

Partial Execution of the KSA

for i = 0 to 255 **do**
  $S_o[i] = i$
$j_0 = 0$
for i = 0 to 255 **do**
  $j_{i+1} = (j_i + S_i[i] + K[i])$
  $S_{i+1}[i] = S_i[j_{i+1}]$
  $S_{i+1}[j_{i+1}] = S_i[i]$

Swap $S[i], S[j]$

- After j_{L+1} is set, KSA performs swap($S[i], S[j]$)
- Suppose $S[j_{L+1}] = Q$
- Then Q is swapped to position L
- Why it it useful??

Partial Execution of the KSA

Swap $S[i], S[j]$

for i = 0 to 255 **do**
  $S_o[i] = i$
$j_0 = 0$
for i = 0 to 255 **do**
  $j_{i+1} = (j_i + S_i[i] + K[i])$
  $S_{i+1}[i] = S_i[j_{i+1}]$
  $S_{i+1}[j_{i+1}] = S_i[i]$

- Consider when no other swaps have perturbed j_{L+1}
- Then, $Q = j_{L+1}$ and $S[i] = Q = j_{L+1}$
- If we could reveal the byte $S[L]$, we might learn j_{L+1}
- We could solve the previous equation $K[L] = j_{L+1} - j_L - S_L[L]$

  i.e. the next byte of the key!!

Partial Execution of the KSA

- No perturbation of j_{L+1} and the ability to look at $S[L]$ seem to be unlikely
- This occur with some statistical significance (FMS is a statistical attack)
- With Weak IVs, these conditions are met with around 5% probability
- Let us consider the weak $IV = (L, n-1, x)$, where:
  - $\boldsymbol{L}$ is the byte we are trying to find
  - $\boldsymbol{n} = 256$
  - **x** is a wildcard (can be any value)

Weak IV=(3,255,x)

for i = 0 to 255 **do**
$j = (j + S_i[i] + K[i])$
swap$(S[i] = S[j])$

0	1	2	3	4	5	....	255

i, j

$$j = j + S[i] + K[i]$$
$$j = 0 + 0 + 3$$

3	1	2	0	4	5	....	255

i $\qquad$ j

Weak IV=(3,255,x)

for i = 0 to 255 **do**
$j = (j + S_i[i] + K[i])$
swap$(S[i] = S[j])$

3	1	2	0	4	5	....	255
	i		j				

$$j = j + S[i] + K[i]$$
$$j = (3 + 1 + 255)\ mod\ 256 = 3$$

3	0	2	1	4	5	....	255
	i		j				

Weak IV=(3,255,x)

for i = 0 to 255 **do**
$j = (j + S_i[i] + K[i])$
swap($S[i] = S[j]$)

3	0	2	1	4	5	....	255

i j

$$j = j + S[i] + K[i]$$
$$j = (3 + 2 + x)\ mod\ 256 = x'$$

3	0	x'	1	4	5	....	255

i j

Weak IV=(3,255,x)

for i = 0 to 255 **do**
$j = (j + S_i[i] + K[i])$
swap($S[i] = S[j]$)

3	**0**	**x'**	**1**	**4**	**5**	**....**	**255**

i $\qquad\qquad\qquad\qquad$ j

$$j = j + S[i] + K[i]$$
$$j = x' + 1 + K[i] = Q$$

3	**0**	**x'**	**Q**	**4**	**5**	**....**	**1**		**255**

i $\qquad\qquad\qquad\qquad$ j

Weak IV=(3,255,x)

for i = 0 to 255 **do**
$j = (j + S_i[i] + K[i])$
swap$(S[i] = S[j])$

3	0	x’	Q	4	5	....	1		255
			i				j		

- Assume that S[0], S[1], S[3] remain unchanged for the rest of the KSA
- What is the first byte of KS output by the PRGA?

Weak IV=(3,255,x)

- Assume that S[0], S[1], S[3] remain unchanged for the rest of the KSA

while *output* **do**
$i = (i + 1)\ mod256$
$j = (j + S[i])\ mod256$
$\text{swap}(S[i], S[j])$
$\text{output } S[(S[i] + S[j])\ mod256]$

3	**0**	**x’**	**Q**	**4**	**5**	**....**	**1**		**255**

$$i = 1$$
$$j = j + S[i] = 0 + S[1] = 0$$

$KS[0]$ output: $S\big[S[i] + S[j]\big] = S[S[1] + S[0]] = S[0 + 3] = S[3] = \boldsymbol{Q}$

- With 5% probability, $Q = j_{L+1}$
- Allowing us to solve $K[L] = j_L + 1 - j_L - S_L[L]$
- Problem: we cannot observe the keystream, so we observe the ciphertext

Obtaining RC4 Keystream

- Every 802.11 frame is wrapped in a SNAP header
- First byte of the LLC/SNAP is `0xAA`
- The next 7 bytes are also static and known.
- Thus:

$$M[0] \oplus KS[0] = C[0]$$
$$KS[0] = C[0] \oplus M[0]$$

In Summary

To determine the L-th byte of the RC4 Key

- Run the KSA for L steps
- Capture packet with weak IV of type (L, n-1, x)
- Obtain $KS[L]$
- With 5% probability $K'[L] = j_{L+1} - j_L - S_L[L]$
- Choose $K[L]$ to be the most frequenet candidate among the found $K'[L]$
- Reiterate until key recovered