Applied Computer Sciences and Biosciences

RC4 Algorithm

Quentin Stickler, B.Sc.

April 9, 2024

Agenda

The Rise and Fall of RC4

Why it's not really used anymore

- Stream cipher with variable key-size length
- Used to be most wiedely used stream cipher in Software applications
- Invented in 1987 by Ron Rivest for RSA security
- Kept secret but got leaked in 1994
- Easy to implement and quite fast
- ...but also very vulnerable

RC4 Algorithm

How does it work?

- Consists of two parts
- Part 1: Initialization
- Part 2: Keystream Generator
- S-Box (Array) with length of 256
- Two 8-byte sized counters i and j

RC4 Initialization

Part One: Filling S-Box and T-Box

- S-Box with length 256
- Counters i and j set to 0
- Linear filling of the S-Box from 0 to 255 (S[0] = 0, S[1] = 1…)
- Following loop will be run:

```
for x in range(256):    ###Initilaze S-Box and T-Box
S[x] = x
T[x] = asciikey[x % keylength]
```

Initialization

Example

- Text = "TestText"
- Key = "TestKey"
- S-Box = [0, 1, 2, 3 . . . , 255]
- Initialization of T-Box:
  - Keylength = 7
  - Ascii-Text = 84 101 115 116 75 101 121

84	101	115	116	75	101	121
84	101	115	116	75	101	121
. . .	. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	84	101	115	116

Attacking RC4 in WEP

- Lorem ipsum

Permutation Example

- S-Box Initialization:

0	1	2	3	4	5	6
...	...	...	...	...	...	...
249	250	251	252	253	254	255

- i = 0
- j = (j + S[i] + T[i]) mod(256)
- j = (84 + 0 + 84) mod(256) = 168 mod (256) = 168
- Swap S[i] (0) and S[j] (84)
- S[i] = 84, S[j] = 0

Permutation Example Cont'd

84	1	2	3	4	5	6
...	...	...	...	...	...	...
80	81	82	83	0	85	86
...	...	...	...	...	...	...
249	250	251	252	253	254	255

- i = 1
- j = (j + S[i] + T[i]) mod(256)
- j = (186 + 1 + 101) mod(256) = 288 mod (256) = 32
- Swap S[i] (1) and S[j] (186)
- S[i] = 186, S[j] = 1

Permutation Example Cont'd

84	186	2	3	4	5	6
...	...	...	...	...	...	...
80	81	82	83	0	85	86
...	...	...	...	...	...	...
249	250	251	252	253	254	255

- i = 2
- j = (j + S[i]) + T[i] mod(256)
- j = (47 + 2 + 115) mod(256) = 126 mod (256) = 126
- Swap S[i] (1) and S[j] (47)
- S[i] = 47, S[j] = 2

Permutation Example

Final S-Box Form

84	186	47	208	12	95	222	212	71	9	26	246	103	38	28	165
138	68	130	10	50	143	72	155	39	139	112	16	79	78	196	146
216	179	159	178	34	119	59	56	63	183	53	197	100	236	101	4
176	250	116	67	5	60	194	35	105	87	118	218	97	168	1	77
44	229	25	48	141	42	175	91	94	211	121	169	215	89	99	24
98	164	181	129	255	185	110	8	220	154	109	219	201	153	120	62
51	0	217	37	20	226	43	127	170	227	243	249	133	126	161	156
82	167	140	115	145	74	182	83	184	104	189	81	52	233	172	245
157	66	124	177	102	80	147	171	106	162	70	30	199	6	69	18
173	45	32	88	125	221	7	65	75	158	232	128	237	190	108	248
13	144	2	46	49	31	134	123	92	40	114	254	131	213	41	93
117	253	23	137	234	209	224	136	107	90	202	223	132	27	15	207
73	195	239	64	206	251	149	228	231	166	187	214	86	242	191	76
192	58	142	61	57	193	33	244	180	205	111	3	122	36	22	14
240	252	238	188	247	85	203	174	200	11	148	152	160	230	210	29
96	235	163	150	17	204	54	55	198	151	225	21	135	113	19	241

- Result = Permutated S-Box
- All numbers from 0-255 in "random" places

Keystream Generator

- Generate keystream depending on length of given plaintext

```
        for x in range(plaintextlength):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        currentValue = S[i]
        S[i] = S[j]
        S[j] = currentValue
        t = (S[i] + S[j]) % 256
        keystream.append(S[t])
        return keystream
```

Keystream Generator

Example, i = 0

- i = 0, j = 0
- i = (0 + 1) mod 256 = 1
- j = (0 + 186) mod 256 = 186 mod 256 = 186
- Swap S[i] (186) and S[j] (202)
- t = (202 + 186) mod 256 = 388 mod 256 = 132
- keystream = [132,]

Keystream Generator

Example, i = 1

- i = 1, j = 186
- i = (1 + 1) mod 256 = 2
- j = (186 + 47) mod 256 = 233 mod 256 = 233
- Swap S[i] (47) and S[j] (11)
- t = (47 + 11) mod 256 = 58 mod 256 = 58
- keystream = [132, 58,]

Keystream Generator

Example, i = 2

- i = 2, j = 233
- i = (2 + 1) mod 256 = 3
- j = (233 + 208) mod 256 = 451 mod 256 = 185
- Swap S[i] (208) and S[j] (90)
- t = (208 + 90) mod 256 = 298 mod 256 = 42
- keystream = [132, 58, 42, ...]
- Final keystream = [132, 58, 42, 7, 129, 233, 245, 149]

Encryption

- Plaintext XOR keystream
- Plaintext = "TestText" = [84, 101, 115, 116, 84, 101, 120, 116]
- Binary: 01010100 01100101 01110011 01110100 01010100 01100101 01111000 01110100
- Keystream = [132, 58, 42, 7, 129, 233, 245, 149]
- = 10000100 0111010 00101010 0000111 10000001 11101001 11110101 10010101
- 01010100 01100101 01110011 01110100 01010100 01100101 01111000 01110100 XOR
- 10000100 00111010 00101010 00000111 10000001 11101001 11110101 10010101 =
- 11010000 01011111 01011001 01110011 11010101 10001100 10001101 11100001

Decryption

- Ciphertext XOR keystream
- Plaintext = "TestText"
- Binary: 01010100 01100101 01110011 01110100 01010100 01100101 01111000 01110100
- Keystream = [132, 58, 42, 7, 129, 233, 245, 149] =
- 10000100 0111010 00101010 0000111 10000001 11101001 11110101 10010101
- 01010100 01100101 01110011 01110100 01010100 01100101 01111000 01110100 XOR
- 10000100 00111010 00101010 00000111 10000001 11101001 11110101 10010101 =
- 11010000 01011111 01011001 01110011 11010101 10001100 10001101 11100001

Attacks on RC4

- Lorem ipsum

WEP

Short summary

- Lorem ipsum

Attacking RC4 in WEP

- Lorem ipsum

Recovery of unknwon bytes

Theorem

Let K_n be the RC4 key value at position n. Let IV_n be a tuple of $(n, N-1, V)$, where $N = 256$ and $V \in 0, \ldots, 255, n \geq 3$ and k_n the known keystreambyte at position n. Then $K_n = k_n - \sum_1^n x - V - (\sum_3^n K_n)$

- How many IVs are sufficient to determine K_n?
- Determine probability that S_0, S_1, S_3 remain unchanged

Prevention against this attack

- Lorem ipsum