Foundations of Network and Computer Security

John Black

Lecture #24
Nov 23rd 2004

CSCI 6268/TLEN 5831, Fall 2004

Announcements

Proj #2 – Due today

- Can hand in Tuesday if need be

- Quiz #4: next time

- No class Thurs (Thanksgiving) or Tues (the 30^{th})

WEP and RC4

- We saw last time how WEP uses RC4
  - $C = P \oplus RC4(v, k)$
  - v is a 24-bit IV
  - k is a 40-bit key (could be 104-bits too)
- C is then sent along with v
  - So we seed RC4 with 64 bits, 24 of which are public and 40 of which are private
- Turns out this is bad

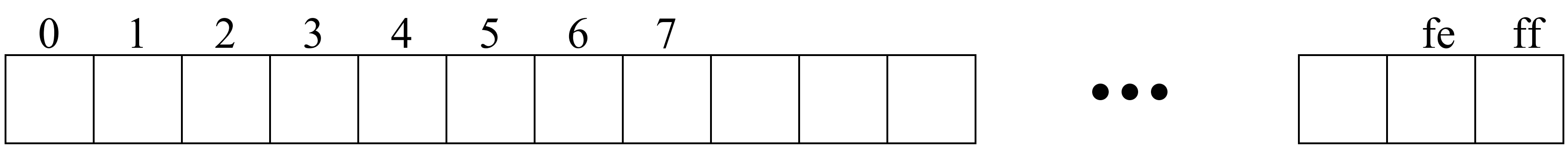
RC4

- Designed by Rivest
- Secret algorithm (trade secret) for years
  - One day, reverse-engineered code showed up on a cypherpunks mailing list (1995)
  - Was called “alleged RC4” for a while
  - Now just assumed to be RC4
- Meant to be simple enough to be memorized
  - This was to circumvent export problems
- Most common mode used in SSL
  - But they don’t use it like WEP does

RC4 Algorithm

- Uses an internal state of:
  - 256 byte permutation S
  - 2 pointers into that permutation
- So how many states possible?
  - $256! * 256^2 \approx 2^{1700}$
  - Exhaustive search on the state is clearly not a good idea

State array S



Byte values; must be a permutation of {0,…,255}

RC4 Algorithm (cont)

- Two phases of the algorithm:
  - Key Schedule Algorithm (KSA)
    - Digest the key to get the initial state
  - Pad Generation (PRGA)
    - Start generating endless stream of pseudorandom bytes
- We run the KSA first with the key; discard the key; keep the state; then ask the PRGA for bytes as-needed

RC4-KSA(K)

- K has 8 bytes for 24-bit IV v and 40-bit WEP key k

```
RC4_KSA(K)
for i ← 0 to 255
       S[i] ← i
j ← 0
for i ← 0 to 255
       j ← (j + S[i] + K[i mod 8])
       S[i] ↔ S[j]
```

- Assume all arithmetic is mod 256 when manipulating indices to S (so j stays in the range 0 to 255)
- Please take a moment to memorize this one

RC4-PRGA

```
i ← 0; j ← 0
while (1) do
     i++
     j ← (j+S[i])
     S[i] ↔ S[j]
     output S[S[i] + S[j]]
```

- State S is global; PRGA outputs bytes forever
- Once again, assume mod 256 as needed

Let’s Run KSA

- First we need an IV v and WEP key k
  - eg, v=0x020441, k=0x0567f1a3dd

RC4 Key K

0	1	2	3	4	5	6	7
02	04	41	05	67	f1	a3	dd

IV v   WEP Key k

- Next we initialize the state array
  - Permutation is the identity, i and j point to 0

0	1	2	3	4	5	6	7				…		fe	ff
0	1	2	3	4	5	6	7				• • •		fe	ff

i, *j* → 0

Running the KSA

	0	1	2	3	4	5	6	7				…		fe	ff
S_0	0	1	2	3	4	5	6	7				…		fe	ff

$i \leftarrow 0 \qquad j \leftarrow 0 + 0 + K[0] = 2$

	0	1	2	3	4	5	6	7				…		fe	ff
S_1	2	1	0	3	4	5	6	7				…		fe	ff

$i \leftarrow 1 \qquad j \leftarrow 2 + 1 + K[1] = 11$

	0	1	2	3	4	5	6	7				…		fe	ff
S_2	2	7	0	3	4	5	6	1				…		fe	ff

$i \leftarrow 2 \qquad j \leftarrow 2 + 1 + K[1] = 7$

S_i denotes the state of array S after i iterations of the for loop

Then Run PRGA

- Suppose we have some permutation at the end of the KSA; now run PRGA once

0	1	2	3	4	5	6	7						fe	ff
2	7	0	fa	a0	9	22	9a				•••		b1	5f

- i is 1, j is 0
- j ← j + S[i], so j ← S[1]
- swap S[i] and S[j]
- output S[S[i] + S[j]], so S[S[1] + S[S[1]]]
- So first byte of PRGA after KSA is run is
  S[S[1] + S[S[1]]]

The Attack

- David Wagner noticed in 1996 that if the first bytes of the RC4 seed were of a certain form, interesting things happen
- Fluhrer, Mantin, Shamir expanded on this and showed how it applies to WEP
- Idea:
  - Suppose WEP IV is of the following form:
    - 0x03ffxx
    - Here, “xx” means any value, so we just need it to start with 0x03ff

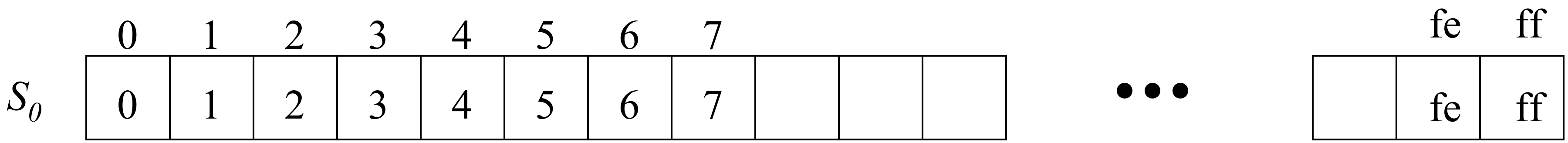
Special IV

- What does K look like then?

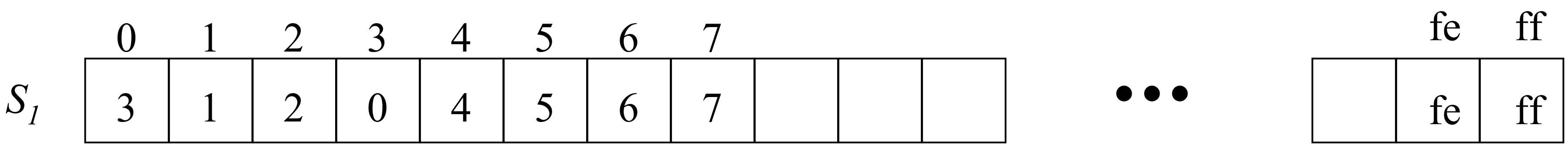
RC4 Key K	0	1	2	3	4	5	6	7
	03	ff	xx	T				
		IV v				WEP Key k		

- T here is the first part of the WEP key k
  - This is a secret value that we would like to find
  - The IV is of course public, so we can recognize when it is in this special form
- Let's run the KSA with this IV

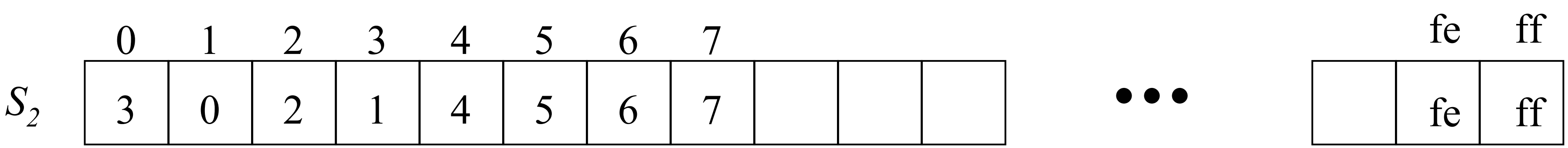
K = 03 ff xx T … ;   Set X = xx



i ← 0   j ← 0 + 0 + K[0] = 3



i ← 1   j ← 3 + 1 + K[1] = 3



i ← 2   j ← 3 + 2 + K[2] = 5 + X

Note: We can compute all of this without the WEP key k

	0	1	2	3	4	5	6	7				X+5		fe	ff
S_3	3	0	S_2[X+5]	1	4	5	6	7			• • •	2	• • •	fe	ff

i ← 3 ***j ← (X+5) + 1 + K[3] = X+6+T***

	0	1	2	3	4	5		X+5		fe	ff
S_4	3	0	S_2[X+5]	S_3[X+6+T]	4	5	• • •	2	• • •	fe	ff

Note: This is the first time something happens that we cannot compute using just the IV

- Now let's assume that for the remaining 252 iterations of i (from 4 through 255) we never disturb S[0], S[1], or S[3]
  - i will never point here again, but j might; we assume that it won't, and see what happens
- Then the PRGA runs and outputs S[S[1]+S[S[1]]] as its first byte
  - This is S[0 + S[0]] = S[3] = S_3[X+6+T]
  - We can solve for T

Solving for T

- We have $S_3[X+6+T]$ and we know S_3 completely
  - Search for $S_3[X+6+T]$ in the S_3 array
  - Say the index is Z
  - Then T = Z-X-6
    - Taken mod 256, as always
  - This gives T, the first byte of the secret WEP key k
    - Other bytes found in similar manner

But what if j messes things up?

- Recall j was "randomly" jumping through S
  - It may point to 0, 1, or 3 and then our computation doesn't work
    - Moreover, if j messes things up, we can't detect that it did or didn't!
  - Let's assume j is uniform and random
    - What is the probability it will avoid these 3 locations?
    - $(1\text{-}3/255)^{252} \approx \lim_{N \to \infty} (1\text{-}3/N)^N = e^{-3} \approx 0.05$
    - So 95% of the time, j messes us up…

Using statistics

- However, 5% of the time we strike gold
  - And we can assume that the times we don't, we get a uniformly random value out
  - Imagine a die with 256 sides with 1 side coming up 5% of the time and the other sides coming up 0.37% of the time
    - So 1 side is 13 times more likely than any other
  - Run a statistical test:
    - For χ different values of X, get the candidate T
    - The majority element is the correct T value
      - Calculations show that $\chi = 60$ is sufficient
      - Challenge problem #4: compute how many values of X are needed to get a probability p that the majority element is T

Example

- Suppose you have IV's:
  - 0x03ff00, 0x03ff01, 0x03ff02, …, 0x03ff3b
  - You get candidate T values (using our computation):
    - 12 19 21 217 19 204 1 7 19 22 49 57 52 250 111 19 18 7 20 …
    - So we choose T=19 as the first byte of the secret key

RC4 Attack: Further Notes

- Need IV's of a certain form
  - A passive attacker has to wait until they occur naturally
  - An active attacker will just set them to whatever values he needs

- Note that we need to get the first byte of the key stream
  - With WEP this is easy because the first byte of every frame is 0xaa
    - Has to do with an encapsulation standard

Extending the Attack

- Once you get T = K[3] you can get all further bytes as well
  - For 40-bit WEP, there are 4 more bytes
  - For 104-bit WEP there are 12 more
    - So for this attack, key size DOES matter
  - To attack K[4] you need to have K[3]
    - It had better be right!
    - IVs should be of the form 0x04ffxx

Practical Implications

- An attacker has to sit outside and collect a LOT of packets to get your WEP key
- This attack, combined with the BGW attack from last time are quite damaging, but it still makes sense to run WEP if you're worried about securing your network
- AirSnort and other programs have the FMS attack built in