

Different attacks on the RC4 stream cipher

Andreas Klein

Ghent University
Dept. of Pure Mathematics and Computer Algebra
Krijgslaan 281 - S22
9000 Ghent
Belgium

Overview

The RC4 algorithm

Overview

The RC4 algorithm

Wireless LAN

Overview

The RC4 algorithm

Wireless LAN

Distinguishing Attacks

Overview

The RC4 algorithm

Wireless LAN

Distinguishing Attacks

Fortuitous states

Overview

The RC4 algorithm

Wireless LAN

Distinguishing Attacks

Fortuitous states

Weaknesses in the key scheduling phase

The FMS-attack

Overview

The RC4 algorithm

Wireless LAN

Distinguishing Attacks

Fortuitous states

Weaknesses in the key scheduling phase

The FMS-attack

A new attack

A correlation in the RC4 pseudo random generator

The basic version of the attack

Using only few sessions

Optimisation for WEP

Overview

The RC4 algorithm

Wireless LAN

Distinguishing Attacks

Fortuitous states

Weaknesses in the key scheduling phase

The FMS-attack

A new attack

A correlation in the RC4 pseudo random generator

The basic version of the attack

Using only few sessions

Optimisation for WEP

RC4A

The RC4 algorithm

RC4 key scheduling

- 1: {initialization}
- 2: **for** i **from** 0 **to** $n - 1$ **do**
- 3: $S[i] := i$
- 4: **end for**
- 5: $j := 0$
- 6: {generate a random permutation}
- 7: **for** i **from** 0 **to** $n - 1$ **do**
- 8: $j := (j + S[i] + K[i \bmod l]) \bmod n$
- 9: Swap $S[i]$ and $S[j]$
- 10: **end for**

The RC4 algorithm

RC4 pseudo random generator

- 1: {initialization}
- 2: $i := 0$
- 3: $j := 0$
- 4: {generate pseudo random sequence}
- 5: **loop**
- 6: $i := (i + 1) \bmod n$
- 7: $j := (j + S[i]) \bmod n$
- 8: Swap $S[i]$ and $S[j]$
- 9: $k := (S[i] + S[j]) \bmod n$
- 10: **print** $S[k]$
- 11: **end loop**

Wireless LAN

→ Wireless LAN encrypts the packages with RC4.

Wireless LAN

- Wireless LAN encrypts the packages with RC4.
- The older protocol WEP uses keys of the form initialisation vector|main key.
- The newer protocol WPA uses a *temporal key hash* to compute the session key.

Wireless LAN

- Wireless LAN encrypts the packages with RC4.
- The older protocol WEP uses keys of the form initialisation vector|main key.
- The newer protocol WPA uses a *temporal key hash* to compute the session key.
- WEP is broken.

Errors in WEP

- The main key has only 5 bytes. (Corrected in later implementations.)

Errors in WEP

- The main key has only 5 bytes. (Corrected in later implementations.)
- The initialisation vector consists of only 3 bytes. This leads to a birthday attack.

Errors in WEP

- The main key has only 5 bytes. (Corrected in later implementations.)
- The initialisation vector consists of only 3 bytes. This leads to a birthday attack.
- The protocol adds CRC-checksums to the packages before encrypting. This is the wrong order.

Errors in WEP

- The main key has only 5 bytes. (Corrected in later implementations.)
- The initialisation vector consists of only 3 bytes. This leads to a birthday attack.
- The protocol adds CRC-checksums to the packages before encrypting. This is the wrong order.
- Public known header informations are encrypted. This leads to known plain text attacks without an enhancement of security.

Errors in WEP

- The main key has only 5 bytes. (Corrected in later implementations.)
- The initialisation vector consists of only 3 bytes. This leads to a birthday attack.
- The protocol adds CRC-checksums to the packages before encrypting. This is the wrong order.
- Public known header informations are encrypted. This leads to known plain text attacks without an enhancement of security.
- People use wireless LAN where they could use standard LAN, even in high security areas.

Distinguishing Attacks

Goal

Distinguish the RC4 pseudo random sequence from a true random sequence.

Distinguishing Attacks

Goal

Distinguish the RC4 pseudo random sequence from a true random sequence.

- Golić 1997: The sum of the last bits at time step t and $t + 2$ is correlated to 1, $\approx 2^{40}$ bytes of the RC4 pseudo random sequence are distinguishable from a true random sequence.

Distinguishing Attacks

Goal

Distinguish the RC4 pseudo random sequence from a true random sequence.

- Golić 1997: The sum of the last bits at time step t and $t + 2$ is correlated to 1, $\approx 2^{40}$ bytes of the RC4 pseudo random sequence are distinguishable from a true random sequence.
- Fluhrer, McGrew 2000: Correlations between two successive output bytes, $\approx 2^{30}$ bytes are sufficient to distinguish RC4 from random noise.

Fortuitous states

Defintion

An RC4 state (S-Box, i and j) in which only m consecutive S-Box elements are known and only those m elements participate in producing the next m successive outputs, is call a fortuitous state of length m .

Fortuitous states

Defintion

An RC4 state (S-Box, i and j) in which only m consecutive S-Box elements are known and only those m elements participate in producing the next m successive outputs, is call a fortuitous state of length m .

length m	2	3	4	5	6
number of fortuitous states	516	290	6540	25419	101819

Fortuitous states

Defintion

An RC4 state (S-Box, i and j) in which only m consecutive S-Box elements are known and only those m elements participate in producing the next m successive outputs, is call a fortuitous state of length m .

length m	2	3	4	5	6
number of fortuitous states	516	290	6540	25419	101819

- If we observe the output of a fortuitous state, we know the corresponding S-Box entries with probability $\frac{1}{n}$.

Fortuitous states

Defintion

An RC4 state (S-Box, i and j) in which only m consecutive S-Box elements are known and only those m elements participate in producing the next m successive outputs, is call a fortuitous state of length m .

length m	2	3	4	5	6
number of fortuitous states	516	290	6540	25419	101819

- If we observe the output of a fortuitous state, we know the corresponding S-Box entries with probability $\frac{1}{n}$.
- Predictive states and non-fortuitous states are generalisation of that concept.

Fortuitous states

Defintion

An RC4 state (S-Box, i and j) in which only m consecutive S-Box elements are known and only those m elements participate in producing the next m successive outputs, is call a fortuitous state of length m .

length m	2	3	4	5	6
number of fortuitous states	516	290	6540	25419	101819

- If we observe the output of a fortuitous state, we know the corresponding S-Box entries with probability $\frac{1}{n}$.
- Predictive states and non-fortuitous states are generalisation of that concept.
- No practical attack using fortuitous states is known.

Weaknesses in the key scheduling phase

Weaknesses in the key scheduling phase

- There are 256^{256} possible keys of full length, but only $256!$ different S-Box states. Since $256!$ does not divide 256^{256} , we know that the distribution of the initial S-Box permutation must differ from the uniform distribution.

Weaknesses in the key scheduling phase

- There are 256^{256} possible keys of full length, but only $256!$ different S-Box states. Since $256!$ does not divide 256^{256} , we know that the distribution of the initial S-Box permutation must differ from the uniform distribution.
- Mironov 2002: The identity is the most likely initial permutation and the cycle $(1, 2, \dots, n - 1, 0)$ is the most unlikely initial permutation.

Weaknesses in the key scheduling phase

- There are 256^{256} possible keys of full length, but only $256!$ different S-Box states. Since $256!$ does not divide 256^{256} , we know that the distribution of the initial S-Box permutation must differ from the uniform distribution.
- Mironov 2002: The identity is the most likely initial permutation and the cycle $(1, 2, \dots, n - 1, 0)$ is the most unlikely initial permutation.
- Suggestion: Do not use the first $12 \cdot 256$ bytes of the RC4 pseudo random sequence to avoid a possible exploitation of this weakness.

The FMS-attack

The FMS-attack

- The basic version of the FMS-attack assumes session keys of the form initialisation vector|main key. The initialisation vector should be 3 bytes long.

The FMS-attack

- The basic version of the FMS-attack assumes session keys of the form initialisation vector|main key. The initialisation vector should be 3 bytes long.
- We consider only initialisation vectors of the form $(3, 255, X)$.

The FMS-attack

- The basic version of the FMS-attack assumes session keys of the form initialisation vector|main key. The initialisation vector should be 3 bytes long.
- We consider only initialisation vectors of the form $(3, 255, X)$.
- In the first step of the key scheduling, j is set to 3 and S-Box has the value $4, 1, 2, 0, 4, 5, \dots, 255$.

The FMS-attack

- The basic version of the FMS-attack assumes session keys of the form initialisation vector|main key. The initialisation vector should be 3 bytes long.
- We consider only initialisation vectors of the form $(3, 255, X)$.
- In the first step of the key scheduling, j is set to 3 and S-Box has the value $4, 1, 2, 0, 4, 5, \dots, 255$.
- In the next step, i is increased to 1 and j is increased by $S[1] + K[1] = 1 + 255 \equiv 0 \pmod{256}$. The S-box has the value $4, 0, 2, 1, 4, 5, \dots, 255$.

The FMS-attack

- The basic version of the FMS-attack assumes session keys of the form initialisation vector|main key. The initialisation vector should be 3 bytes long.
- We consider only initialisation vectors of the form $(3, 255, X)$.
- In the first step of the key scheduling, j is set to 3 and S-Box has the value $4, 1, 2, 0, 4, 5, \dots, 255$.
- In the next step, i is increased to 1 and j is increased by $S[1] + K[1] = 1 + 255 \equiv 0 \pmod{256}$. The S-box has the value $4, 0, 2, 1, 4, 5, \dots, 255$.
- The value X is known so we can compute the third step of the key scheduling.

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .
- The probability that $S[0]$, $S[1]$ and $S[3]$ are not changed in the remaining 252 steps of the key scheduling is approximately $(1 - \frac{3}{256})^{252} \approx \frac{1}{e^3} \approx 0.05$.

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .
- The probability that $S[0]$, $S[1]$ and $S[3]$ are not changed in the remaining 252 steps of the key scheduling is approximately $(1 - \frac{3}{256})^{252} \approx \frac{1}{e^3} \approx 0.05$.
- With probability > 0.05 we observe t as first byte of the RC4 pseudo-random sequence.

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .
- The probability that $S[0]$, $S[1]$ and $S[3]$ are not changed in the remaining 252 steps of the key scheduling is approximately $(1 - \frac{3}{256})^{252} \approx \frac{1}{e^3} \approx 0.05$.
- With probability > 0.05 we observe t as first byte of the RC4 pseudo-random sequence.
- About 60 sessions of the $(3, 255, X)$ are sufficient to reconstruct the first key byte K .

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .
- The probability that $S[0]$, $S[1]$ and $S[3]$ are not changed in the remaining 252 steps of the key scheduling is approximately $(1 - \frac{3}{256})^{252} \approx \frac{1}{e^3} \approx 0.05$.
- With probability > 0.05 we observe t as first byte of the RC4 pseudo-random sequence.
- About 60 sessions of the $(3, 255, X)$ are sufficient to reconstruct the first key byte K .
- Once K is known, we can treat it as part of the initialisation vector and reconstruct the other key bytes.

The FMS-attack

- We can express the value t of $S[3]$ after the fourth step of the key scheduling as a function in X and the unknown key byte K .
- The probability that $S[0]$, $S[1]$ and $S[3]$ are not changed in the remaining 252 steps of the key scheduling is approximately $(1 - \frac{3}{256})^{252} \approx \frac{1}{e^3} \approx 0.05$.
- With probability > 0.05 we observe t as first byte of the RC4 pseudo-random sequence.
- About 60 sessions of the $(3, 255, X)$ are sufficient to reconstruct the first key byte K .
- Once K is known, we can treat it as part of the initialisation vector and reconstruct the other key bytes.
- The FMS attack needs between 1000000 and 4000000 sessions to reconstruct the main key.

The FMS-attack (Conclusions)

Advice

Do not use the first bytes of the RC4 pseudo random sequence.

Advice

Do not rely on the RC4 key scheduling to protect your main-key.
Use session keys of the form

`hash-function(session-id|main-key).`

A correlation in the RC4 pseudo random generator

Theorem

Assume that the internal states are uniformly distributed. Then for a fixed public pointer i , we have:

$$P(S[j] + S[k] \equiv i \pmod{n}) = \frac{2}{n} \quad (1)$$

For $c \not\equiv i \pmod{n}$ we have:

$$P(S[j] + S[k] \equiv c \pmod{n}) = \frac{n-2}{n(n-1)} \quad (2)$$

A correlation in the RC4 pseudo random generator

Theorem

Assume that the internal states are uniformly distributed. Then for a fixed public pointer i , we have:

$$P(S[j] + S[k] \equiv i \pmod{n}) = \frac{2}{n} \quad (1)$$

For $c \not\equiv i \pmod{n}$ we have:

$$P(S[j] + S[k] \equiv c \pmod{n}) = \frac{n-2}{n(n-1)} \quad (2)$$

Proof (sketch):

- Use $k \equiv S[j] + S[i] \pmod{n}$ to write $S[j] + S[k] \equiv i \pmod{n}$ as $k + S[k] \equiv i + S[i] \pmod{n}$.
- Count the corresponding states.

The basic version of the attack

The basic version of the attack

- Assume that the session key has the form
main key|initialisation vector.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.
- Express the value t of $S[1]$ after the second step of the key scheduling phase as function in $K[0]$ and $K[1]$.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.
- Express the value t of $S[1]$ after the second step of the key scheduling phase as function in $K[0]$ and $K[1]$.
- The probability that $S[1]$ is not changed in the remaining step of the key scheduling is $\approx (1 - \frac{1}{256})^{254} \approx \frac{1}{e}$.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.
- Express the value t of $S[1]$ after the second step of the key scheduling phase as function in $K[0]$ and $K[1]$.
- The probability that $S[1]$ is not changed in the remaining step of the key scheduling is $\approx (1 - \frac{1}{256})^{254} \approx \frac{1}{e}$.
- The first step of the pseudo random generator sets j to $S[1]$.

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.
- Express the value t of $S[1]$ after the second step of the key scheduling phase as function in $K[0]$ and $K[1]$.
- The probability that $S[1]$ is not changed in the remaining step of the key scheduling is $\approx (1 - \frac{1}{256})^{254} \approx \frac{1}{e}$.
- The first step of the pseudo random generator sets j to $S[1]$.
- Apply the theorem to conclude:

$$P(t \equiv 1 - S[k] \pmod n) \approx \frac{1}{e} \cdot \frac{2}{n} + (1 - \frac{1}{e}) \cdot \frac{n-2}{n(n-1)} \approx \frac{1.36}{n}.$$

The basic version of the attack

- Assume that the session key has the form main key|initialisation vector.
- Guess the first key byte $K[0]$.
- Compute the first step of the key scheduling phase.
- Express the value t of $S[1]$ after the second step of the key scheduling phase as function in $K[0]$ and $K[1]$.
- The probability that $S[1]$ is not changed in the remaining step of the key scheduling is $\approx (1 - \frac{1}{256})^{254} \approx \frac{1}{e}$.
- The first step of the pseudo random generator sets j to $S[1]$.
- Apply the theorem to conclude:

$$P(t \equiv 1 - S[k] \pmod n) \approx \frac{1}{e} \cdot \frac{2}{n} + (1 - \frac{1}{e}) \cdot \frac{n-2}{n(n-1)} \approx \frac{1.36}{n}.$$

- We must observe about 25,000 sessions to recover $K[1]$.

Variants

- Once the byte $K[1]$ is known, we can attack the bytes $K[2]$ and so on.

Variants

- Once the byte $K[1]$ is known, we can attack the bytes $K[2]$ and so on.
- The session keys are not really independent from each other. For a main key of length b bytes, it is possible that j is set to 1 at a time step between 1 and b . If this happens, the basic attack will fail to recover the key byte $K[1]$.
- It is possible to cope with such ugly keys.

Variants

- Once the byte $K[1]$ is known, we can attack the bytes $K[2]$ and so on.
- The session keys are not really independent from each other. For a main key of length b bytes, it is possible that j is set to 1 at a time step between 1 and b . If this happens, the basic attack will fail to recover the key byte $K[1]$.
- It is possible to cope with such ugly keys.
- One can use the attack also against session keys of the form initialisation vector|main key.

Variants

- Once the byte $K[1]$ is known, we can attack the bytes $K[2]$ and so on.
- The session keys are not really independent from each other. For a main key of length b bytes, it is possible that j is set to 1 at a time step between 1 and b . If this happens, the basic attack will fail to recover the key byte $K[1]$.
- It is possible to cope with such ugly keys.
- One can use the attack also against session keys of the form initialisation vector|main key.
- Combining the idea of this attack with the idea of weak initialisation vectors, we get an attack which does not use the first 256 bytes of the RC4 pseudo random sequence.

Reducing the number of sessions

Problem

We are not able to observe enough (25000) session.

Reducing the number of sessions

Problem

We are not able to observe enough (25000) session.

- Consider the following case:
 - ▶ Key length : 128 bits (16 bytes).
 - ▶ Number of sessions: ≈ 12000 .

Reducing the number of sessions

Problem

We are not able to observe enough (25000) session.

- Consider the following case:
 - ▶ Key length : 128 bits (16 bytes).
 - ▶ Number of sessions: ≈ 12000 .
- Use the observe sessions to calculate the a posteriori probability for $t = f(K[0], K[1])$. The a posteriori probability is given by

$$P_i = P(t = i \mid \text{the absolute frequencies are } f_j) = \frac{\prod_{j=1}^n p_{i,j}^{f_j}}{\sum_{k=1}^n \prod_{j=1}^n p_{k,j}^{f_j}}$$

with $p_{i,j} = p = \frac{1.36}{n}$ for $i = j$ and $p_{i,j} = q = \frac{1-p}{n-1}$ for $i \neq j$.

Reducing the number of sessions (continued)

→ Compute the a posteriori probabilities for the other key bytes.

Reducing the number of sessions (continued)

- Compute the a posteriori probabilities for the other key bytes.
- The a posteriori entropy is about 64.

Reducing the number of sessions (continued)

- Compute the a posteriori probabilities for the other key bytes.
- The a posteriori entropy is about 64.
- Start a complete key search, but test the keys with the highest a posteriori probability first.
- You can expect to find the right key after 2^{64} steps. For comparison a full key search needs 2^{128} steps.

Optimisation for WEP

Problem

It is easy to observe extra Wireless LAN Packages (use ARP-spoofing). But it is time consuming to store observed sessions.

Optimisation for WEP

Problem

It is easy to observe extra Wireless LAN Packages (use ARP-spoofing). But it is time consuming to store observed sessions.

- E. Tews, R. Weinmann and A. Pyshkin (Darmstadt) found a way to attack the different key bytes in parallel.
- They use

$$t = S_3^{-1}((3 + i) - X(i + 2)) - j_3 - \sum_{j=3}^{i+3} S_3(j)$$

for an estimation of $K[i]$. (S_3 and j_3 denote the S-box and j after the third step of the key scheduling phase.)

- The approximation is wrong for a small fraction of the key space. For strong keys, we must still recover the key bytes sequentially.

Definition of RC4A

RC4A pseudo random generator

- 1: {initialization}
- 2: $i := 0$
- 3: $j_1 := 0$ $j_2 := 0$
- 4: {generate pseudo random sequence}
- 5: **loop**
- 6: $i := (i + 1) \bmod n$
- 7: $j_1 := (j_1 + S_1[i]) \bmod n$
- 8: Swap $S_1[i]$ and $S_1[j_1]$
- 9: $k_2 := (S_1[i] + S_1[j_1]) \bmod n$
- 10: **print** $S_2[k_2]$
- 11: $j_2 := (j_2 + S_2[i]) \bmod n$
- 12: Swap $S_2[i]$ and $S_2[j_2]$
- 13: $k_1 := (S_2[i] + S_2[j_2]) \bmod n$
- 14: **print** $S_1[k_1]$
- 15: **end loop**

Attacking RC4A

Theorem

Assume that all permutations have the same probability, and that S_1 and S_2 are independent. Then:

$$P(S_1[j_1] + S_1[k_1] + S_2[j_2] + S_2[k_2] \equiv 2i \pmod n) = \frac{1}{n-1}. \quad (3)$$

Attacking RC4A

Theorem

Assume that all permutations have the same probability, and that S_1 and S_2 are independent. Then:

$$P(S_1[j_1] + S_1[k_1] + S_2[j_2] + S_2[k_2] \equiv 2i \pmod n) = \frac{1}{n-1}. \quad (3)$$

→ The correlation is weaker than the corresponding correlation of RC4.

Attacking RC4A

Theorem

Assume that all permutations have the same probability, and that S_1 and S_2 are independent. Then:

$$P(S_1[j_1] + S_1[k_1] + S_2[j_2] + S_2[k_2] \equiv 2i \pmod n) = \frac{1}{n-1}. \quad (3)$$

- The correlation is weaker than the corresponding correlation of RC4.
- But we can still mount an attack on this correlation.