# Network Security Class
# Lab Session 4

Stefano Zanella - 621796

July 2013

# 1 Abstract

This experience aims at experimenting the different attack techniques provided by the **Aircrack-ng** tool against the **WEP** protocol.

In particular, we'll try to crack a WEP key by applying the three different mechanisms provided by aircrack and see how they compare to each other in terms of reliability and performance.

Before detailing the attack implementation, we'll give an overview of the different techniques available in Aircrack; then, we'll detail the experiment and provide some brief analysis over the results.

# 2 Aircrack

Aircrack is a set of tools for auditing wireless networks. Included in the set there are some command line utilites to sniff wireless traffic and to crack WEP keys from sniffed packets.

## Sniffing wireless packets

The basic building block for carrying out an attack on WEP is to capture as much traffic as possible. This is a relatively straightforward task, since all commercial WiFi NICs can work in **promiscuous mode**; that is, they can bypass layer 2 filtering of packets by MAC address, and pass everything on the upper layers for subsequent elaboration.

Aircrack relies on NICs promisc mode and provides a tool, `airodump-ng`, to actually collect traffic and log it to an output file.

## FMS/Korek method

This attack takes his name from Fluhrer, Mantin and Shamir, who depicted it in their 2001 paper *"Weaknesses in the Key Scheduling Algorithm of RC4"*, enhanced by the attacks discovered by the person that goes under the pseudonym of Korek, who published them on a public Internet forum.

The original FMS attack relied on the fact that IVs are public and the same pre-shared key is reused many times with many different IVs. It was

based on the discovery of particular IVs that let secret key bytes leak directly into the pseudo-random key stream. Also, it leverages on the fact that, given the structure of the 802.11 payload format, an attacker knows the first byte of each plaintext payload, and hence can derive the first byte of RC4 keystream output. Under these assumptions, they proved that with roughly 60 IVs suitable to retrieve a particular key byte, the probability of correcly deriving such key byte was greater than 0.5. Hence, since in WEP IVs are just 3 bytes long and usually generated with a counter, and since pre-shared keys are also quite short, this attack leads to a high success probability with a much lower time complexity than a bruteforce attack.

However, they also provided a simple way to mitigate this attack's effectiveness: since it works just on the first pair of bytes generated by each IV, they suggested to simply discard these first bytes before actually outputting anything, and also to obviously discard known weak IVs - even though they also showed a similar attack related to invariance weakness which works with any IV and yields a $O(2^{40})$ complexity instead of the $O(2^{16})$ of the weak IVs attack, which is still much lower than the $O(2^{256})$ required for bruteforcing typical RC4 with 32 bytes keys.

The seventeen Korek attacks are designed around the main FMS discoveries, and basically evolve around the "Invariant Weakness" FMS attack. They all are probabilistic attacks, and can be grouped in three subsets:

- the first 8 attacks try to guess key bytes based on the first output byte of the key stream, each relying on IVs that satisfy different conditions;
- the second 8 attacks include the knowledge of the first and second output byte of the key stream
- the last attack is a reverse method to reduce the size of the search space, also known as "inverted attack". It depicts four conditions which when met allow to reject certain values of key bytes.

Aircrack incorporates all these attacks in a series of statistical tests that handle each byte of the key individually. Given each attack has a different success probability, usually of  5implemented, so at the end of the statistical analysis each key byte will have a list of possible candidates, and the one with more votes is selected. The resulting key is then tested and if it isn't the correct one, then a bruteforce phase is entered, which explores key bytes with lower probability.
The search space extension can be tuned by setting the **fudge factor**; that is, the user can set how many bytes to bruteforce starting from the initial guessed key.

Clearly, this factor has roughly an inverse relationship with the number of useful packets captured, since the most they are, the higher is the probability that the FMS/Korek method can determine the right key without the need of bruteforcing additional keys.

## PTW method

The PTW attack was presented in 2007 by Physkin, Tews and Weinmann. It has a similar structure to that of the FMS/Korek attack, but relies on two new concepts that allowed to obtain a success probability greater than 50sniffed packets, instead of the much greater 4,000,000 to 6,000,000 estimate for the FMS/Korek attack.

The first concept this attack relies upon is a new correlation function discovered by Klein in 2005 that removes all the constraints on the internal state of RC4 or keystream needed by FMS and Korek, thus allowing to make use of every sniffed packet to perform key recovery.

The second concept is a change in the attack structure. Physkin, Tews and Weinmann discovered additional correlations between multiple bytes of the RC4 key and the keystream, so that the voting mechanism in PTW is not performed on a byte-per-byte basis anymore, but instead uses the multibyte correlation to vote on *13 partial sums of consecutive key bytes*, and then performing simple subtactions on these partial sums to set *all key bytes*. Thus, if the key is not correct, an update in one of the values of these partial sums can be reflected in a change in many key bytes with just 12 subtractions.

It should be noted that Aircrack implementation of the PTW attack is divided in two phases: the first phase uses just ARP packets for decryption of the keystream, while the second phase uses all captured packets (discarding some well-known packets that confuse the PTW algorithm).
Basically, the need for ARP packets is due to their much predictable structure. In fact, the first 16 bytes of an ARP packet (both for requests and replies) are constant. This allows to directly obtain 16 bytes of the keystream by simply XORing this 16 bytes pattern to the first 16 bytes of the enciphered packet. Also, an encrypted ARP packet sent over a wireless medium is prepended with a 28 byte plaintext header which allows for easy discrimination of ARP packets over other protocols (the destination address being `FF:FF:FF:FF:FF:FF`).

### Bruteforce method

This is an experimental attack which, despite the name, isn't a plain bruteforce attack.
Basically, it still performs an initial FMS/Korek statistical analysis (narrower than the actual FMS/Korek algorithm), then tries to bruteforce the key by exploring the search space around the bytes that received most votes by the FMS calculations.

## 3 Attack Implementation

To actually carry out an attack in lab environment, we used two distinct machines: one impersonating the attacker, and one impersonating a legitimate user (so to generate traffic to be dumped). Both machines were running a Linux environment, so all commands that follow are based on this single requirement.

Also, to reduce time needed to actually obtain some result, WEP was configured with a 40 bit key instead of 104.

### Retrieving network information

To start, we need to identify some basic information about the wireless network we want to break: in particular, to sniff traffic with aircrack we need to know the radio channel used by the network and the MAC address of the AP.
This can be done easily since it's part of the basic (legitimate) functionality of a wireless network card.

We start by bringing up the wireless interface:

```
ifconfig wlan0 up
```

At this point, given we know the SSID of the network (`ap1` in our case), we can retrieve needed information with a single command:

```
iwlist wlan0 scanning essid ap1
```

Among other information, we get what we need:

- Channel: **11**

- AP MAC address: **00:25:86:F8:0E:D5**

## Sniffing packets

At this point we are ready to sniff packets, but we first need to generate traffic to be sniffed. Due time constraints, we have forced the legitimate machine to generate ARP traffic by running a loop in which an ARP request was made and then the ARP table cleared:

```
while :; do
  ping -c 1 192.168.11.1
  arp -d 192.168.11.1
done
```

On the other machine, we first put the wireless NIC into monitoring mode:

```
airmon-ng start wlan0 channel 11
```

Then started dumping packets into a `.cap` file for subsequent use:

```
airodump-ng -c 11 --bssid 00:25:86:f8:0e:d5 -w ap1traffic wlan0
```

## Key recovery

We let `airodump` continue to retrieve packets while trying the different attacks to recover the key.

We first tried the experimental bruteforce method:

```
aircrack-ng -y ap1traffic*.cap
```

Then the FMS/KoreK attack:

```
aircrack-ng -K ap1traffic*.cap
```

And then the (default) PTW attack:

```
aircrack-ng ap1traffic*.cap
```

Since while trying to recover the key we also continued to dump traffic, we tried all attacks with different amount of packets/IVs so to see how much data they need to successfully recover the key and how they perform with respect to each other in different conditions.

# 4 Attacks comparison

- Bruteforce attack: it failed at first with just 12,500 IVs available, but have been successful and really fast when feeded with 20,000 IVs. In this case, it took time comparable to that of the PTW attack to recover the correct key (both in terms of absolute time spent running and average key trials performed). Feeding it with more IVs didn't seem to improve performance considerably (though no precise measurement was done to this purpose).
- FMS/KoreK attack: this, as expected, was the method that took longer and had the most demanding requirements in term of dumped traffic. We let it run for more than 10 minutes with just 14,000 IVs, and had to stop manually since it wasn't able to neither recover the key, nor give us a failure message.
  The situation slightly improved with more IVs, in the sense that we were able to receive a failure message after few minutes stating that the attack couldn't be carried out. To actually obtain the correct key we needed to feed it with 180,000 IVs, plus we needed to manually set the desired key length. In that case the FMS/KoreK attack run in a time comparable to that of PTW and bruteforce attacks, but clearly dumping that amount of traffic took much longer.
- PTW attack: it worked with even 14,000 IVs, but strangely it didn't seem to be faster than the single bruteforce attack. It showed slightly better performance when fed with roughly 30,000 IVs, in the sense that it recovered the correct key faster than bruteforce on average, and almost always needed to try less keys than it (something between 5 and 200).

In conclusion, it seems that the bruteforce method isn't just a simple addition to the FMS/KoreK attack, but rather a massive improvement over it (and maybe the word *bruteforce* is not very suited to describe it) with performance near to that of the PTW method. Unfortunately, there isn't much documentation over this method, neither in Aircrack website, nor in the Aircrack source code. All considerations about it were made by looking at the source code and comparing it with that of the full FMS/KoreK attack.

# 5 References


1. Weaknesses in the Key Scheduling Algorithm of RC4
   Scott R. Fluhrer, Itsik Mantin, Adi Shamir
   Selected Areas in Cryptography 2001

2. Break WEP Faster with Statistical Analysis
   Rafik Chaabouni
   School of Computer and Communication Sciences Semester Project 2006

3. Practical attacks against WEP and WPA
   Martin Beck, Erik Tews
   TU-Dresden, TU-Darmstadt 2008

4. Aircrack official website
   http://www.aircrack-ng.org