



# Cryptography E2006

*Insecurity of the IEEE 802.11 Wired Equivalent Privacy (WEP) standard*

**Lasse Kosetski Deleuran**

*20030587, ld@daimi.au.dk*

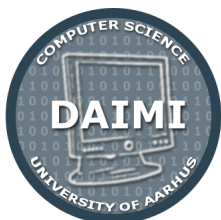
**Thomas Loftager Nielsen**

*20030594, tln@daimi.au.dk*

**Henrik Gammelmark**

*20031899, geemark@daimi.au.dk*

December 15, 2006



---

Department of Computer Science, University of Aarhus  
Aabogade 34, DK-8200 Aarhus N

<http://www.daimi.au.dk>

---

## Abstract

The purpose of this report is to investigate some of the known vulnerabilities of the Wired Equivalent Privacy (WEP) protocol, which is part of the IEEE 802.11b standard for wireless network connectivity.

We wish to demonstrate some of these weaknesses and attacks on the protocol, and thus display why it should not be used for security-critical networks.

We do not mean to give a very deep mathematical understanding of a single issue, but rather give a more complete overview of the problems with the WEP standard. We will, however, go to some depth with the most revolutionary attack.

Also the algorithms used in WEP will be described only briefly to the extent required to understand the attacks.

# Contents

|           |                                     |           |
|-----------|-------------------------------------|-----------|
| <b>I</b>  | <b>Introduction and research</b>    | <b>5</b>  |
| <b>1</b>  | <b>Descriptions of algorithms</b>   | <b>6</b>  |
| 1.1       | RC4 . . . . .                       | 6         |
| 1.2       | CRC-32 . . . . .                    | 7         |
| 1.3       | WEP protocol . . . . .              | 8         |
| <b>II</b> | <b>Known attacks</b>                | <b>11</b> |
| <b>2</b>  | <b>Overview</b>                     | <b>12</b> |
| 2.1       | Brute force . . . . .               | 12        |
| 2.2       | FMS . . . . .                       | 13        |
| 2.3       | KoreK attacks . . . . .             | 13        |
| 2.4       | Integrity Attack . . . . .          | 13        |
| 2.5       | Chopchop . . . . .                  | 14        |
| 2.6       | IV collisions . . . . .             | 14        |
| <b>3</b>  | <b>Details of the FMS attack</b>    | <b>16</b> |
| 3.1       | Weak IVs . . . . .                  | 16        |
| 3.2       | Exploiting KSA and PRGA . . . . .   | 17        |
| 3.3       | Number of weak IVs needed . . . . . | 18        |
| <b>4</b>  | <b>Combining the attacks</b>        | <b>19</b> |
| 4.1       | Generating traffic . . . . .        | 19        |
| 4.2       | Software for cracking WEP . . . . . | 19        |
| 4.3       | Time consumption . . . . .          | 20        |

|   |           |
|---|-----------|
| <b>III Conclusion</b>                   | <b>21</b> |
| <b>5 Conclusion</b>                     | <b>22</b> |
| 5.1 State of WEP . . . . .              | 22        |
| 5.2 Alternatives and remedies . . . . . | 22        |
| <b>Bibliography</b>                     | <b>24</b> |
| <b>IV Appendix</b>                      | <b>25</b> |
| <b>A Linearity of CRC</b>               | <b>26</b> |

**Part I**

**Introduction and research**

# Chapter 1

## Descriptions of algorithms

WEP makes use of the algorithms RC4 and CRC-32. We will give a brief introduction to these before going into the details about how they are used in WEP.

### 1.1 RC4

RC4 is a stream cipher, meaning it will generate an endless stream of pseudo-random bits, called a keystream, given an initial key. To encrypt a message, the keystream will then be XOR'ed with the plaintext as if it was a genuine one time pad. To perform decryption the RC4 generator is restarted with the same secret key, and the keystream is XOR'ed with the ciphertext to reproduce the message.

#### 1.1.1 Building blocks

The internal state of RC4 consists of a permutation  $S$  of 256 bytes and indices  $i$  and  $j$  into this array. RC4 consists of two separate algorithms - A key scheduling algorithm (KSA) and a pseudo-random generation algorithm (PRGA).

Note that all calculations of indexes ( $i, j$  etc) into the array  $S$  of the following two algorithms is performed modulo  $|S|$ , which is 256 in WEP.

##### Key Scheduling Algorithm

Initially the key scheduling algorithm (see figure 1.1) is executed. It initializes the permutation  $S$  by setting the  $i$ 'th entry of  $S$  to  $i$  and scrambling  $S$  afterwards by using the secret key  $K$ .

##### Pseudo-Random Generation Algorithm

After being initialized by the KSA, RC4 is ready to generate an endless stream of bytes using the PRGA algorithm (see figure 1.2). It does so by incrementing  $i$  and changing  $j$  pseudo-randomly and swapping elements of  $S$ . We are guaranteed that at least once every 256 iterations, all values of  $S$  have been swapped at least once. To generate the pseudo-random bitstream, the loop is executed as many times as required to generate the wanted number of bytes.

Note that the PRGA loop is very similar to the KSA loop; in fact the only difference is the lack of key-byte introduction in the PRGA.

```

KSA(key)
  // Initialization of S permutation
  for i from 0 to 255
    S[i] := i
  j := 0

  // Scramble using the secret key
  for i from 0 to 255
    j := j + S[i] + key[i mod |key|] mod 256
    swap(S[i], S[j])

```

Figure 1.1: RC4 KSA algorithm

```

PRGA()
  i := 0
  j := 0

  while NeedMoreBytes
    i := i + 1 mod 256
    j := j + S[i] mod 256
    swap(S[i], S[j])
    output S[S[i] + S[j] mod 256]

```

Figure 1.2: RC4 PRGA algorithm

## 1.2 CRC-32

Cyclic Redundancy Check (CRC) is a kind of hash function, that given a message produces a short digest (checksum) that can be used to verify the integrity of the data, detect transmission errors etc. This is done by calculating the CRC on the received message at the receiver side and check for equality with the CRC that was sent along with the message. CRC-32 produces 32-bit checksums.

CRC is mathematically defined as the remainder  $R$  after division with the key  $K$  in the field  $\mathbb{Z}_2[X]/\langle p \rangle$ <sup>1</sup>, where all elements and  $p$  are polynomials in  $\mathbb{Z}_2[X]$ <sup>2</sup>:

$$M X^n = Q K + R$$

where  $n$  is the degree of  $K$  and  $M$  is the message to be verified. The checksum to be send on the network is  $M X^n - R$ , so the receiver only has to check that  $(M X^n - R)/K = 0$  to verify the message.

The beauty of this representation is the simplicity of calculations when using binary representation. A polynomial can be represented with a single bit for each quotient. The multiplication with  $X^n$  simply adds  $n$  zeros to the binary representation of  $M$  and because addition equals subtraction in the field,  $M X^n - R$  is written as the bits from  $M$  followed by the bits from  $R$ . The division is carried out by bitwise XOR'ing, so the generation of CRCs can be implemented as shown in figure 1.3.

<sup>1</sup>See [Lau05, chapter 4] for how to work with polynomials

<sup>2</sup>We have choosen to denote polynomials on the form  $P$  instead of  $P(X)$ , which is the normal convension, in order to make it easier to understand for people who have not followed the course of Algebra

```

int CRC(BitVector M)
  int crc := 0
  for each bit in M
    crc <<= 1      // Shift left
    crc |= bit     // Bring current data bit onto lsb of crc
    if msb(crc) = 1 // crc is divisible by the polynomial
      crc ^= p     // p is the polynomial defining the field
  return crc

```

Figure 1.3: CRC-32 algorithm

For CRC-32 the polynomial  $p$  is

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$$

for which the quotients correspond to  $0x04C11DB7$  in hexadecimal notation.

An important property of the algorithm is the linearity with respect to XOR:

$$CRC(a) \oplus CRC(b) = CRC(a \oplus b)$$

The proof of this is found in appendix A.

There are implementations of the algorithm using tables and other tricks to speed up the process. Some initialize  $crc$  to something different from 0, but these variations preserve the property of linearity and thus do not change the attacks we will present.

## 1.3 WEP protocol

Wired Equivalent Privacy (WEP) is part of the IEEE 802.11 standard ([IEEE99]) of 1999. As the name suggests, the purpose of the protocol is to provide the same level of security as a wired network does. More specifically, nodes not part of the network are unable to use the network or eavesdrop on the communication, while nodes internal to the network are free to use and eavesdrop on the shared media.

### 1.3.1 Secrecy

The Wired Equivalent Privacy protocol is based on the RC4 stream cipher (described in section 1.1) to generate key streams that are XOR'ed with the plaintext being encrypted.

The original key size used by WEP for the RC4 cipher is 64 bits, but larger key sizes of 128 and 256 are also supported.

### Initialization vectors

The effective key size is reduced by 24 bits because 3 bytes are randomly chosen<sup>3</sup> while encrypting each WEP packet and send with the packet unencrypted.<sup>4</sup> This 3-byte value is known as an Initialization Vector (IV), and its purpose is twofold:

- To add some nondeterminism to the encryption process. The IV is sent as plaintext as part of the packet, so the receiver (and any adversary) can easily extract it, and it should therefore not be considered part of the key per se. Without the IVs, a message send twice would always give the same ciphertext and leak information by doing so.
- The RC4 output is basically used as a one-time pad, and hence each key stream should be used only once. By using IVs, the exact same PRGA key stream is not used every time.

### Encryption process

The simple encryption process is depicted in figure 1.4. Decryption is the exact same proces, only the IV is read from the plaintext-part of the packet, instead of generated.

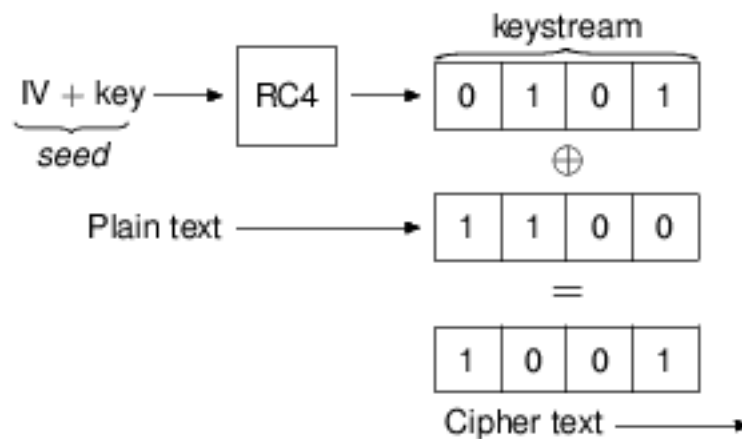


Figure 1.4: WEP encryption process. Image source: [Wik06, article: Wired Equivalent Privacy]

### 1.3.2 Integrity

Appended to the message as part of the plaintext before encryption, is a checksum of the message being sent. This is to ensure against transmission failures and tampering with the message contents. The checksum being used is CRC-32, described in section 1.2.

<sup>3</sup>Some implementations in fact simply use forthcoming numbers

<sup>4</sup>Due to this, the most common WEP key sizes are in practice 40 or 104 bits long

### 1.3.3 Authentication

WEP provides an authentication protocol<sup>5</sup> that basically works as an overcomplicated dynamic MAC<sup>6</sup> address filter.

The two standard WEP authentication methods are:

- *Shared Key Authentication.*  
The access point presents the client with a 128 bit challenge and an IV, which the client can only answer correctly with knowledge of the shared secret key by returning the encryption of the challenge. If the access point agrees with the encryption, the client MAC is white-listed, and it can then send an association request to the access point and thus "connect" to the network.
- *Open System Authentication.*  
In order for a client to authenticate with the access point, it simply needs to supply the same SSID<sup>7</sup> as the access point is using (or the string "ANY").

### 1.3.4 Packet format

The WEP packet format ([Hul02]) is depicted in figure 1.5. The 802.11 header contains information such as source and destination MAC addresses. We won't elaborate further on other 802.11 header parts, as they are irrelevant for this report.

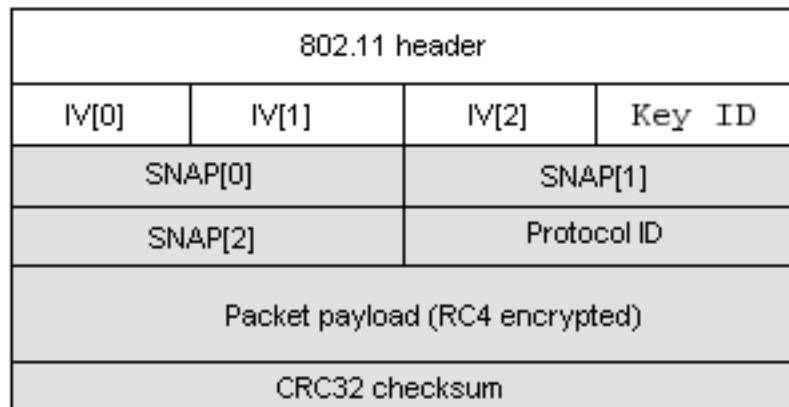


Figure 1.5: WEP packet format: Each line is 32 bits. The encrypted part is grayed.

Everything following the Key ID is encrypted. This is depicted as the shaded part in the illustration.

Special attention should be paid to the Sub-Network Access Protocol (SNAP) header, especially the first byte, which has a constant value of  $0xAA$  (decimal 170)<sup>8</sup>. This is the first byte to be encrypted under each (IV, secret key)-pair, and thus allows for a known-plaintext attack, which will be described in chapter 3.

Also notice, that the length  $|K|$  of the secret key is not specified anywhere in the packet, as this is only present as implicit knowledge to the legitimate users of the network.

<sup>5</sup>[Inc05], [JP88]

<sup>6</sup>In this report, MAC refers to Media Access Control, the physical addressing scheme of 802.11, not Message Authentication Code

<sup>7</sup>Service Set Identifier, a text-string of up to 32 bytes identifying the access point

<sup>8</sup>This is at least true for ARP and IP packets, see [JP88]

## Part II

# Known attacks

# Chapter 2

## Overview

### 2.1 Brute force

The most common and easily implemented attack is using brute force to determine the encryption key. Often, this is not practical due to the huge size of the keyspace, but in the case of 40-bit WEP keys, this is only  $2^{40} = 1,099,511,627,776$  possible keys, and hence a simple brute force attack is indeed doable; especially if the adversary is utilizing specialized hardware.

The attack can be carried out passively. Only a single (arbitrary) packet needs to be captured. We already know a byte of the plaintext because of the SNAP header, and if that byte matches, we can go on and decrypt the entire packet. If we decrypt and get garbage<sup>1</sup>, we simply continue the process.

Using a regular PC, assisted by a cluster of 15 smaller CPUs, it is possible to test 135,000,000 keys every second<sup>2</sup>. A quick calculation reveals that to brute force a 40bit WEP key, will take this computer only two hours and 16 minutes in the worst case. On the average case, the brute force completes within half that time!

Of course this will take considerable more time using only a regular workstation, but it illustrates that brute-forcing the key is possible with not too expensive hardware, that will all fit within a regular PC tower.

#### 2.1.1 Spurious keys

We may still hit a spurious key<sup>3</sup>, that in fact decrypts the packet to something meaningful, but the probability of doing so decreases exponentially with the length of the packet.

The probability that the first byte decrypts fine is  $\frac{1}{256}$ . However, the probability that the entire packet decrypts to something syntactically correct, using a spurious key, is negligible. Even if this should occur it is quickly observed when decryption of other packets fails.

The observation about spurious keys applies to all the attacks presented in the report.

---

<sup>1</sup>the CRC-32 most likely doesn't fit the data if the wrong key is used

<sup>2</sup>2 x 2.8 GHz Pentium4 and 15 x Pico E-12,[Hul06]

<sup>3</sup>As defined in [Sti06, section 2.6]

## 2.2 FMS

The RC4 Key Scheduling Algorithm (KSA) has some special characteristics that can be exploited in a clever way when some specific "weak" IVs are used. Using this famous and highly effective attack, it is possible to recover any WEP key in time linear in the key size.

The attack is described at a glance in [Fog02], and will be discussed in more detail in chapter 3.

## 2.3 KoreK attacks

In 2004, an underground hacker under the alias of *KoreK* published<sup>4</sup> 17 rather sophisticated statistical attacks on WEP, that combined with the FMS attack yields an incredibly effective algorithm.

Unfortunately, all that was published was 188 lines of undocumented C code, with no explanation of what is going on. Also, no articles or documentation is to be found. All we know is that the attacks are speeding up the FMS proces considerably.

## 2.4 Integrity Attack

The use of CRC-32 makes it possible to change the message of a WEP packet and update the checksum to match the changed message.

Suppose we are given a packet  $p$ . We know it is constructed as  $p = (m \parallel CRC(m)) \oplus k$ , where  $m$  is the plaintext message,  $CRC$  is the CRC-32 checksum and  $k$  is a key stream.

Let  $m'$  be the message we want to change  $m$  to by flipping some bits. Define the difference  $d = m \oplus m'$ . The linear property of CRC gives us:

$$m \oplus m' = d \Rightarrow CRC(m) \oplus CRC(m') = CRC(d) \Rightarrow CRC(m') = CRC(m) \oplus CRC(d)$$

With this we can see that  $p \oplus (d \parallel CRC(d))$  is a valid packet with the changed message because:

$$\begin{aligned} p \oplus (d \parallel CRC(d)) &= (m \parallel CRC(m)) \oplus k \oplus (d \parallel CRC(d)) \\ &= (m \parallel CRC(m)) \oplus (d \parallel CRC(d)) \oplus k \\ &= ((m \oplus d) \parallel (CRC(m) \oplus CRC(d))) \oplus k \\ &= (m' \parallel CRC(m')) \oplus k \end{aligned}$$

This kind of attack is interesting because it is often possible to guess parts of messages such as http headers, where the IP address of the receiver is located at a certain position within the packet. If the adversary can guess the IP, he can change it to his own. The changed package will then be decrypted by the access point which will send the plaintext to the new IP. The trick is to specify, for instance, an address on the internet controlled by the adversary, causing the packet to be decrypted by the access point, and forwarded to the adversary over the internet.

<sup>4</sup>[Kor04b], a WiFi community forum

## 2.5 Chopchop

The chopchop (due to [Kor04a]) attack allows an adversary to decrypt any given packet one byte at a time (Hollywood style). It is also possible to expand the packet, one byte at a time, thus revealing more of the keystream than was actually used.

### 2.5.1 Decrypting a packet

Chopchop exploits the fact that WEP uses CRC-32 to verify packet integrity.

The attack proceeds as follows: You chop off the last byte of the packet data, yielding an invalid packet. The checksum can be made valid again by XOR'ing with a certain pattern. This pattern depends only on the byte we just chopped off because of the linearity of CRC (see section 2.4).

Now we only have to guess the value of the removed byte. For every guess we compute the effects the value would have had on the checksum, and XOR this with the checksum of the original packet. If our guess is correct, the new packet will be valid and should be accepted by an access point, which will repeat it to the entire network. Otherwise it will be dropped by the access point and we can try with the next guess.

We do not want to wait for a response we may never receive, so we may change the destination MAC address of the 802.11 packet to a pseudo-random one. Now we just need to maintain a table of MAC addresses and corresponding plaintext bytes. This allows us to try all 256 possibilities "at once". We can begin cracking the next byte once we have a hit.

Of course this map is only required if the AP uses a new IV for the re-transmission. If it blindly re-sends the packet with the same IV, we can simply fire all 256 packets, and wait until one is repeated, knowing which character was correct by simple comparison.

It should be mentioned, that in practice not all access points are vulnerable. Some drop short frames, so a total decryption is not directly possible.

### 2.5.2 Expanding the keystream

To expand the keystream, we simply compute the change of the checksum when the plaintext is concatenated with the byte 0. We can now take a guess at the ciphertext byte just like in the last attack and use the access point as an oracle to tell us if our guess was correct. When we have the correct ciphertext byte, we know that this is the same as the byte from the key stream because the plaintext byte is 0.

This expansion will be utilized in an attack found in section 2.6.

## 2.6 IV collisions

IV collisions are bound to occur often due to the fact that the number of possible IVs is limited to  $2^{24}$ . At first glance, 16,7 million combinations might sound like a lot, but because the reverse birthday paradox<sup>5</sup>

---

<sup>5</sup>[Wik06, article: Birthday Paradox]

we will experience an IV collision with probability  $p$  for every  $\sqrt{2 \cdot 2^{24} \ln\left(\frac{1}{1-p}\right)}$  packets observed, assuming the IVs are chosen uniformly at random. More concretely, with a 50% chance, we will observe a collision every 4,822 packets. With a 95% certainty, we will statistically see a collision every 10,025 packets.

Another possibility for collisions arise when some hardware implementations chooses IVs incrementally, and resets its counter to zero when the hardware is initialized. Obviously, this will lead to many collisions as most sessions will send relatively few packets compared to the 16,7 million possibilities.

To actually exploit these collisions, we need the keystream generated from the IV. After obtaining a ciphertext along with its corresponding plaintext, the key stream used for the encryption can be trivially recovered:

$$\text{keystream} = \text{ciphertext} \oplus \text{plaintext}$$

Therefore, if an attacker is able to find the key stream for a given IV, all packets sent using that IV can be decrypted.

The real problem for the attacker is obtaining the plaintext hidden in the ciphertext, but fortunately the chopchop attack from section 2.5 allows us to decrypt any packet (and expand the keystream to any desired length).

To extend this attack, an adversary can construct a table of  $2^{24}$  entries mapping all IVs to their corresponding keystreams resulting in a total memory usage of  $2^{24} \cdot 1,500 = 23.43$  gigabytes. The 1,500 bytes is the MTU of 802.11<sup>6</sup>.

This attack is rather slow in practice, as it requires the use of the chopchop key stream expansion (see section 2.5.2), if we want to be able to decrypt packets of any length. Therefore, recovering all  $2^{24}$  possible keystreams is only a theoretical attack.

---

<sup>6</sup>Most wireless equipment has an Maximum Transfer Unit of 1500 octets

## Chapter 3

# Details of the FMS attack

The purpose of the FMS attack is to extract the secret key one byte at the time by using IVs with a particular format. When we know the byte at position  $B$ , we can use that knowledge to obtain the byte at position  $B + 1$ . Probability theoretical considerations are applied to provide educated guesses for the individual bytes of the key.

This chapter is based primarily on the articles [Fog02] and [SF02]. The details are provided in [SF01], which is the original paper by Scott Fluhrer, Itsik Mantin and Adi Shamir.

### 3.1 Weak IVs

It was discovered that a particular class of Initialization Vectors (IVs) have some interesting properties. The format of these “weak” IVs is depicted in figure 3.1.  $B$  denotes the index of the byte in the secret key that we want to guess<sup>1</sup>.  $\alpha$  can be any value in  $\{0, 1, \dots, 255\}$ . The reason these IVs are weak will become apparent later.

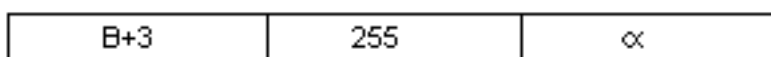


Figure 3.1: Class of weak 24-bit IVs

By simple math, the probability that a packet with a weak IV for a specific  $B$  occur, is  $\frac{2^8}{2^{24}} = 2^{-16}$ , or one in 65,536 packets. Because this is constant for all values of  $B$ , the probability that a packet with *any* usable  $B$  is received, increases with  $|K|$ , thus maintaining the linearity of the attack.

For instance, if  $|K| = 40$  bits (5 bytes), statistically the probability the a given packet contains a weak IV we can use is  $5 \cdot 2^{-16}$  (one in 13,107). On the other hand, if the key size is 104 bits (13 bytes), the chance of a packet containing a usable weak IV is  $13 \cdot 2^{-16}$  (one in 5,041).

<sup>1</sup>Recall that this byte has index  $B + 3$  in the key stream because of the IV

## 3.2 Exploiting KSA and PRGA

When attempting to find the byte at index  $B$  in the secret key, the knowledge of all previous bytes and the bytes of the IV completely determines the first  $B + 3$  iterations of KSA (see figure 1.1). Iteration  $B + 4$  will make the byte to be guessed occur at a specific position which will not change in the rest of the iterations of KSA, with a significant probability. Because of the first plaintext byte being  $\text{SNAP} = 0xAA$  (see section 1.3.4) and the first ciphertext byte  $c$  being known, it is possible to get the first byte of output  $z$  from PRGA, and use this to get the secret key byte.

We will show the details for finding the first byte. The remaining bytes are found similarly and we handle the assumptions to be made for the general case.

### 3.2.1 Obtaining the first byte of the secret key

The first byte ( $B = 0$ ) in the secret key has index  $B + 3 = 3$  in the streaming key, so a usable IV is:

$$(3, 255, \alpha)$$

where  $\alpha$  can be any byte. For clarity we use subscript indices on variables, indicating the last iteration having been run. Before the first iteration, the state array will contain the identity permutation:

$$S_0 = \{0, 1, \dots, 255\}$$

After the first three iterations, the state of KSA is:

$$i_3 = 2, j_3 = 5 + \alpha, S_3[0] = 3, S_3[1] = 0, S_3[2] = 5 + \alpha, S_3[3] = 1, S_3[5 + \alpha] = 2$$

Let  $y = 6 + \alpha + K[3]$ . The state after the fourth iteration is:

$$i_4 = 3, j_4 = 5 + \alpha + S_3[3] + K[3] = y, S_4[0] = 3, S_4[1] = 0, S_4[2] = 5 + \alpha, S_4[3] = y, S_4[5 + \alpha] = 2, S_4[y] = 3$$

Assume that the values of  $S_4[0]$ ,  $S_4[1]$  and  $S_4[3]$  don't change during the remaining iterations of KSA. The probability for this assumption to hold is approximately 5%, which will be shown in the next section.

Assuming these values are still the same in  $S_{256}$ , PRGA will output the first value  $z$  in the end of the first iteration (see figure 1.2). The values of PRGA after this iteration are:

$$i = 1, j = S_{256}[i] = 0, z = S_{256}[S_{256}[i] + S_{256}[j]] = S_{256}[0 + 3] = y$$

We know that WEP XORs the plaintext with the streaming key, so  $c = z \oplus 0xAA$ . Now we have everything needed to extract the secret key byte  $K[3]$ :

$$z = c \oplus 0xAA = y = 6 + \alpha + K[3] \Leftrightarrow K[3] = (c \oplus 0xAA) - 6 - \alpha$$

All of these values are known and the key byte is found, but there are some issues to be taken care of, which we will handle next.

### 3.2.2 Probability for the assumption to hold

The attack is based on the rather bold assumption, that after KSA round  $B + 4$ , the contents of  $S_{B+4}[1]$ ,  $S_{B+4}[S_{B+4}[1]]$  and  $S_{B+4}[B + 3]$ , will not change throughout the execution of KSA.

We know that all the values to the right of  $S[B + 4]$  will be swapped at least once because of the incrementing counter  $i$ , but a value  $S[p]$  to the left of  $S[B + 4]$  (where  $p < B + 4$ ) will only be changed if  $j = p$  in some step. Assuming that  $j$  is effectively random, the probability that  $j$  doesn't become  $p$  in any later step is:

$$\Pr[j \neq p]^{256-(B+4)} = \left(\frac{255}{256}\right)^{252-B}$$

This is a lower bound for the value to be the same in the end of KSA because the value might be swapped back again. We now have the probability that all three values are the same in the end of the last iteration:

$$\Pr[S_{B+4}[1] = S_{256}[1] \wedge S_{B+4}[S_{B+4}[1]] = S_{256}[S_{B+4}[1]] \wedge S_{B+4}[B + 3] = S_{256}[B + 3]] > \left(\left(\frac{255}{256}\right)^{252-B}\right)^3$$

For  $B = 0$ ,  $\left(\left(\frac{255}{256}\right)^{252-B}\right)^3 = 0.0519$ , so there is more than 5% probability for the assumption to be true, and this value increases with larger values of  $B$ .

### 3.2.3 Further assumptions

For the attack to work, we must further assume that swaps in the reproducible steps don't change the values we need. For  $B = 0$  that means  $5 + \alpha$ ,  $\alpha \notin \{0, 1, B + 3\}$ , so there are some restrictions on  $\alpha$ . In general it is easy to see if such situations occur, so this does not pose a real problem.

## 3.3 Number of weak IVs needed

To retrieve a WEP key, we need to obtain a lot of weak IVs to carry out the FMS attack. More precisely, we need at least one packet for each position  $B$  in the secret WEP key, where the first of the above assumptions holds. The assumption holds with probability at least 5% with the remaining 95% giving effectively random outcomes. Therefore we need to observe 60 packets<sup>2</sup> with weak IVs for each  $B \in \{0, \dots, |K| - 1\}$ , to be able to guess the byte with a probability of  $\geq 0.5$ .

For a 40-bit (5 bytes) key, we need to eavesdrop  $5 \cdot 60 = 300$  packets on average with weak IVs, and referring to section 3.1 we know that statistically one in 13,107 packets contains one of these. Assuming a uniform distribution of the IVs, on average we will have to observe  $300 \cdot 13,107 = 3,932,100$  packets. This value is roughly the same when using a 104 bit key (13 bytes):  $(13 \cdot 60) \cdot 5,041 = 3,931,980$ .

---

<sup>2</sup>[SF01, chapter 7.1]

## Chapter 4

# Combining the attacks

### 4.1 Generating traffic

The purpose of generating traffic is gathering as many packets with different IVs as possible. As will become obvious in chapter 3, we can use IVs of a certain format to carry out a key recovery attack.

To generate traffic, the attacker has two options; either to replay previously seen traffic, or to generate it himself. In the first case, he would need to guess correctly that a packet was a request that forced a reply. An example of this is just trying with packets with a frame length of 68 bytes (as these are likely to be ARP <sup>1</sup> requests).

If the desire is to generate new traffic, the adversary has to obtain a valid (key stream, IV pair), which is actually easily done using the chopchop attack.. This enables him to forge any packet he wishes, and transmit it onto the network. As far as the access point is concerned, the packet is perfectly legal and will be treated as such. Thus the attacker just needs to flood the network with requests, so other hosts will respond using new IVs. As technical examples, the attacker could PING a local broadcast address, send ARP requests, or a simple TCP SYN request will solicit a reply from a host. Several of these packets does, however, require knowledge of the IP addresses used by the network, but those were revealed by the chopchop attack as well.

### 4.2 Software for cracking WEP

Various free and/or open source utilities have been developed, that utilize some of the attacks described in this paper. The first generation of these tools implemented the original FMS attack only, but have later been optimized using KoreK's statistical attacks. The most popular package for cracking wep keys is the `aircrack-ng`<sup>2</sup>, which contains all the utilities needed to perform a successful attack.

- `aircrack-ng`  
The tool for the actual offline cracking. It implements the FMS attack along with all 17 KoreK optimizations, and allows to bruteforce the last few keybytes if the first attacks could not determine them.

---

<sup>1</sup>Address Resolution Protocol is used to translate to hardware addresses from IP addresses

<sup>2</sup>The official site is <http://aircrack-ng.org>

- `airdecap-ng`  
Allows decryption of previously captured data, once the key has been determined.
- `aireplay-ng`  
Allows an attacker to generate traffic on the network using several different techniques (Such as chopchop and arp-replay discussed in section 4.1).
- `airodump-ng`  
The tool used to capture data for later cracking. It has several filtering options, so a single network can be targeted.

### 4.3 Time consumption

As the attacks have evolved and become more and more sophisticated, the literature does not agree on this point at all. In the first versions of the attacks, it would be almost infeasible to carry them out in practice, but after the KoreK attacks, among others, it has become is trivial to quickly crack a WEP key using the utilities discussed above.

In a trial test in a controlled enviroment, we managed to crack a 64bit WEP key in less than a second, after having collected 210.000 IVs (10 minutes of eavesdropping), using `aircrack-ng` on a regular laptop.

Another test revealed another 64 bit key in 45 seconds with 250.000 IVs available.

Conclusion

**Part III**

**Conclusion**

# Chapter 5

## Conclusion

### 5.1 State of WEP

As discussed in this report, the problems with WEP basically lies with the choice of tools for the different tasks. Nothing is really done right with regard to confidentiality or authenticity. The ideas and principles behind WEP may be sound and look reasonable on the surface, but the tools are used in insecure ways.

Confidentiality relies on an oversimplified construction of RC4 keys using a secret key and a quite small IVs, which will inevitably lead to related-key attacks.

Authenticity is an important aspect, but it does not work at all in WEP as anyone can spoof a MAC address, which is sent as plaintext.

Message integrity is virtually non-existing, as an algorithm suitable to detect random data corruption, but unsuitable to provide any security against an tampering.

WEP does still provide a level of security sufficient to prevent the casual observer from violating the network, but will fail miserably in no time when faced with an adversary with even the least dedication. The bottom line is: Don't use WEP for anything you wouldn't want the world to see, but when faced with a choice of WEP or nothing, by all means use WEP.

### 5.2 Alternatives and remedies

All of the WEP vulnerabilities have been rectified in the newer WPA standard, so an upgrade to the latter is obviously recommended in any case. Another approach is to completely disregard encryption at the WiFi level and send everything in the clear, while requiring that all users sign on to a VPN<sup>1</sup> solution (like DAIMI does) or similar more reliable technology.

There are several rather simple ways to remove the vulnerabilities discussed in this report:

- *Brute Force*  
Brute force cracking can never be prevented, but it can become infeasible if the network requires the use of very long keys. For instance, WEP2<sup>2</sup>, enforces key sized of 128 bits.

---

<sup>1</sup>Virtual Private Network, based on the IPsec protocol

<sup>2</sup>[Wik06, article: Wired Equivalent Privacy]

- *Dictionary attacks*  
This kind of attack can only be avoided if the end user is not allowed to choose keys, but is given a randomly generated key by the system. This, of course, leads to its own problems with key-prediction.
- *IV collisions*  
 $2^{24}$  Different IVs is far from enough. Increasing the exponent to something like 128 as WEP2 does (which is also the size of the challenges in the authentication protocol of WEP) would render the "keystream table" attack infeasible. However, all hardware should choose IVs in a pseudo-random fashion.
- *FMS*  
There are at least three ways of securing against the FMS attack:
  - Do not use weak IVs. As these can be selected freely by the hardware, there is no reason not to filter them. This is what is done in WEPplus<sup>3</sup>.
  - Do not use the first words of the RC4 keystream, as they are the ones that contain leaked information about the key.
  - Do not simply concatenate key and IV before applying the KSA. Instead a secure hash function taken on the concatenation could be used, as this would remove all statistical correlation.
- *Chopchop*  
By choosing a cryptographically secure hash function instead of the strongly linear CRC-32, this attack is completely nullified.

---

<sup>3</sup>Also known as WEP+. [Wik06, article: Wired Equivalent Privacy]

# Bibliography

- [Fog02] Seth Fogie. Cracking wep. 2002. <http://www.informit.com/articles/article.asp?p=27666>.
- [Hul02] David Hulton. Practical exploitation of rc4 weaknesses in wep environments. 2002. <http://www.dachb0den.com/projects/bsd-airtools.html>.
- [Hul06] David Hulton. Recon 2006 slideshow: Cracking wifi...faster, 2006. <http://openciphers.sourceforge.net/slides/recon-2006.pdf>.
- [IEE99] IEEE. Wireless lan medium access control (mac) and physical layer \*phy() specifications: Higher/speed physical layer extension in the 2.4 ghz band (802.11b). 1999.
- [Inc05] NETGEAR Inc. Wireless networking basics. 2005.
- [JP88] J. Reynolds J. Postel. Rfc 1042 - standard for the transmission of ip datagrams over ieee 802 networks. 1988.
- [Kor04a] KoreK. chopchop (experimental wep attacks), 2004. <http://www.netstumbler.org/showthread.php?t=12489>.
- [Kor04b] KoreK. Netstumbler: Need security pointers, 2004. <http://www.netstumbler.org/showpost.php?p=89036&postcount=11>.
- [Lau05] Niels Lauritzen. *Concrete Abstract Algebra: From Numbers to Groebner Bases*. Press Syndicate of Cambridge University, 2005. ISBN 0521534100.
- [SF01] Adi Shamir Scott Fluhrer, Itsik Mantin. Weaknesses in the key scheduling algorithm of rc4. *SAC2001*, 2001.
- [SF02] Adi Shamir Scott Fluhrer, Itsik Mantin. Attacks on rc4 and wep. 2002.
- [Sti06] Douglas R. Stinson. *Cryptography, Theory and Practice*. Chapman & Hall/CRC, 2006. ISBN 1584885084.
- [Wik06] Wikipedia. Wikipedia: The free encyclopedia, 2006. <http://en.wikipedia.org>.

**Part IV**  
**Appendix**

# Appendix A

## Linearity of CRC

CRC is linear with respect to XOR.

*Proof.* Let  $crc_x$  be the local variable  $crc$  in the algorithm when computing  $CRC(x)$ . Define  $c = a \oplus b$  and  $x[i]$  to be the bit at position  $i$  in  $x$ .

The linearity will be proved by induction by showing that the following invariant is valid throughout the execution of the algorithms:

$$crc_a \oplus crc_b = crc_c$$

The invariant is valid at initialization because  $crc_a = crc_b = crc_c$ .

Suppose that  $crc_a \oplus crc_b = crc_c$ .

The first two operations inside the for-loop maintain validity because:

$$crc_a \oplus crc_b = crc_c \Rightarrow (crc_a \ll 1) \oplus (crc_b \ll 1) = crc_c \ll 1 \text{ and}$$

$$crc_a \oplus crc_b = crc_c \wedge a[i] \oplus b[i] = c[i] \Rightarrow (crc_a \vee a[i]) \oplus (crc_b \vee b[i]) = crc_c \vee c[i] \text{ for all positions } i.$$

At last there are four cases for the values of  $msb(crc_a)$  and  $msb(crc_b)$  to determine the flow after the if-statement:

- $msb(crc_a) = msb(crc_b) = 0$ :  $msb(crc_c) = 0$  so no values are changed.
- $msb(crc_a) = 0 \wedge msb(crc_b) = 1$ :  $msb(crc_c) = 1$ . The invariant is held because  $crc_a \oplus (crc_b \oplus p) = crc_c \oplus p$ .
- $msb(crc_a) = 1 \wedge msb(crc_b) = 0$ : This case is similar to the previous.
- $msb(crc_a) = msb(crc_b) = 1$ :  $msb(crc_c) = 0$ . The invariant is held because  $(crc_a \oplus p) \oplus (crc_b \oplus p) = crc_a \oplus crc_b \oplus p \oplus p = crc_c$ .

It is shown that  $crc_a \oplus crc_b = crc_c$  in the end of the algorithm, and because these are the values to be returned, we have  $CRC(a) \oplus CRC(b) = CRC(a \oplus b)$  and the proof is complete.  $\square$