Weaknesses in WEP and WPA

Martin Guest - Antoine Lefebvre

June 17, 2009

1 Introduction

Over the past ten years, the 802.11 standard, better known as Wifi, has become more and more popular. Therefore, Wi-fi needs a good safety protocol. At the beginning, the WEP was designed to provide a Wired Equivalent Privacy. Wi-fi had to be as secure as the Ethernet. But flaws of the WEP appeared quickly. Nowadays, the WEP is sometimes called the Weak Encryption Protocol.

In 2001, Scott R. Fluhrer, Itsik Mantin and Adi Shamir published the first attack against WEP. After sniffing about 300000 packets in a WEP encrypted network, this attack enables to recover the entire key used by the acces point (AP). Moreover, only a few computations are required. However, only packets using a particular set of Initialisation Vectors (a random datum used to cipher each packet) called weak IVs can be used. And after the attack was published the flaw had been fixed by using "non-weak" IVs.

During the next years, new classes of IVs were discovered and in 2007 a new attack, PTW, was praticable on every IV. Other weaknesses were also exploited like fake authentification or packets injections.

An urgent need to find a new protocol came. WPA, Wi-fi Protected Access, appeared in 2004. It was designed to fill all flaws of WEP. Indeed, it resists all attacks known against the WEP: weak keys exploitation, injection and replay attacks. Moreover, WPA can be implemented on the acual hardware. Indeed, WPA is a transition between WEP and WPA2. It was designed to bring more security with the existing equipment. WPA2 is a protocol based on the stream cipher AES. The efficiency of AES is famous.

In 2004, a person under the pseudonym of KoreK published an attack, called Chopchop, against WEP capable of deciphering any intercepted packet using a low number of packet injections (proportionnal to number of decrypted bytes). Moreover, this attack has a low complexity. Therefore the duration of the attack only depends on the response time of the AP. Another advantage of this attack is that it is working both on WEP and WPA.

The structure of this paper is as follows: in Section 2, we describe technical details on the WEP protocol and on Chopchop. We also give some results to show the efficiency of the attack. In Section 3, we give an overwiew of WPA and known weaknesses - especially concerning Chopchop.

2 Chopchop attack against WEP

2.1 Theory

In order to understand Chopchop attack, we first need to know how WEP works. It is composed of two parts: on the one hand, CRC-32, a function used to detect errors during transmission, adds 4 bytes to the cleartext message. On the other hand , the RC4 algorithm generates a keystream used to cipher the message.

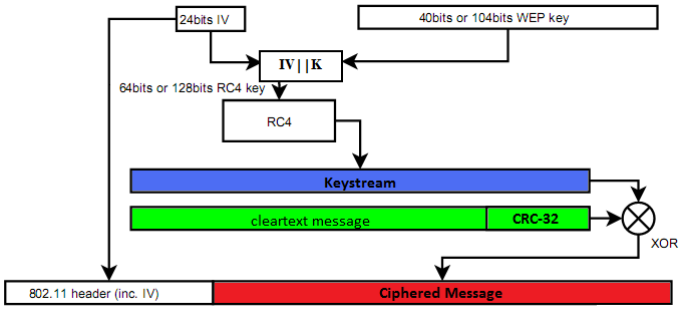


Figure 1: Wep Encryption steps

Each message can be seen as a polynomial P of $\mathbb{F}_2[X]$. The CRC checksum represents the remainder of the division of P by a chosen polynomial (defined by CRC32). That can be computed byte after byte using a table stocking all remainder of each possible byte (see Fig. 2). Consequently, if we add a byte to the message we can deduce the value of the new CRC just from the value of the previous CRC and the value of the additionnal byte. Reciprocally, we can deduce the CRC of a 1-byte troncated message from

the value of the previous CRC and the value of the troncated byte. KoreK showed that by guessing one byte depending on the CRC and the last byte of the cleartext message, we can obtain a 1-Byte troncated ciphered valid message. That valid message will be accepted by the AP during the error detection. Since there are 256 possibilities for 1 single byte, we can deduce the last byte of the message by trying all these 256 values and observe the AP's responses.

```
crc=0;
for(int i=0; i<N; i++)
{
   crc=crc_tab[(crc & 0xff) xor m[i]] xor ( crc >> 8);
}
return crc;
```

Figure 2: CRC algorithm for a message of N bytes [1]

2.2 Algorithm of the attack

Imagine we have intercepted the following encrypted message:

$$\begin{array}{cccccccc} & M_0 & M_1 & ... & M_N & J_3 & J_2 & J_1 & J_0 \\ \oplus & K_0 & K_1 & ... & K_N & K_{N+1} & K_{N+2} & K_{N+3} & K_{N+4} \\ = & R_0 & R_1 & ... & R_N & R_{N+1} & R_{N+2} & R_{N+3} & R_{N+4} \end{array}$$

where $[J_0||J_1||J_2||J_3] = CRC(M_0||...||M_N)$

We want to send a ciphered message S such as:

$$\begin{array}{cccccccc} & M_0 & M_1 & ... & M_N - 1 & I_3 & I_2 & I_1 & I_0 \\ \oplus & K_0 & K_1 & ... & K_{N-1} & K_N & K_{N+1} & K_{N+2} & K_{N+3} \\ = & S_0 & S_1 & ... & S_{N-1} & S_N & S_{N+1} & S_{N+2} & S_{N+3} \end{array}$$

where $[I_0||I_1||I_2||I_3] = CRC(M_0||...||M_{N-1})$

During the computation of the CRC, at the (N+1)-th step of the loop, we have the following relations:

$CRC(M_0...M_N) = CRC\_Tab[(CRC(M_0...M_{N-1})\ \&\ \texttt{0xff})\ \oplus\ M_N]\ \oplus\ (CRC(M_0...M_{N-1}) >> 8)$

$J_0||J_1||J_2||J_3 = CRC\_Tab[I_3\ \oplus\ M_N]\ \oplus\ 0||I_0||I_1||I_2$

Let X be $I_3\ \oplus\ M_N$ and $C_X = CRC\_Tab[X]$. We can deduce all the truncated ciphered message depending on X:

$\forall\ i = 0...N-1,\ S_i = R_i$
$S_N = I_3\ \oplus\ K_N = I_3\ \oplus\ M_N\ \oplus\ M_N\ \oplus\ K_N = X\ \oplus\ R_N$
$S_{N+1} = I_2\ \oplus\ K_{N+1} = I_2\ \oplus\ J_3\ \oplus\ R_{N+1} = C_{X3}\ \oplus\ R_{N+1}$
$S_{N+2} = C_{X2}\ \oplus\ R_{N+2}$
$S_{N+3} = C_{X1}\ \oplus\ R_{N+3}$

Guessing X, we can deduce S. The principle of the attack is as follows: we send $S(X = 0)$ to the AP. If the AP detects an error, we try another value of X. We go on until there is no detected error.

```
X = 0
do {
    send S(X) to the AP ;
    X += 1 ;
}
```

[1] CRC32 is sometimes implemented initialising $crc = \texttt{0xffffffff}$ and returning $crc\ \oplus\ \texttt{0xffffffff}$ but it changes only a little our following equations.

```
while (there is an error)
```

At the end, we have guessed the value of X. Thanks to the previous equations, we also have K_{N+1}.

$$K_{N+1} = J_0 \oplus R_{N+1} = C_{X0} \oplus R_{N+1}$$

We can do N-3 truncations on the same packet to deduce N-3 bytes of the key. It is more than we need. On a wireless network, mainly ARP and IP packets circulate. The first 8 bytes of these packets are known.

0x$AAAA030000000800$ for IP
0x$AAAA030000000806$ for ARP

2.3 Results

We have simulated the attack to check the number of packets we need to find X and to check the complexity. In average, we need 128*m packets to find the last m bytes of one packet.

For each byte to guess, X is uniformly distributed in $[0, 255]$. According to the Central Limit Theorem, the number of packet to send to guess 64 bytes X follows a normal distribution and its average is $64 * 128$. We have obtained a gaussian distribution centered in $64 * 128$ which is coherent (see Figure 3).

To decrypt one N bytes-packet, the AP deciphers $N - 4$ packets, of size $S = 4 \ldots N - 1$. Deciphering takes a linear time functions of S. Therefore, plotting the time functions of the length N of a message, we obtained a parabola (see Figure 4).

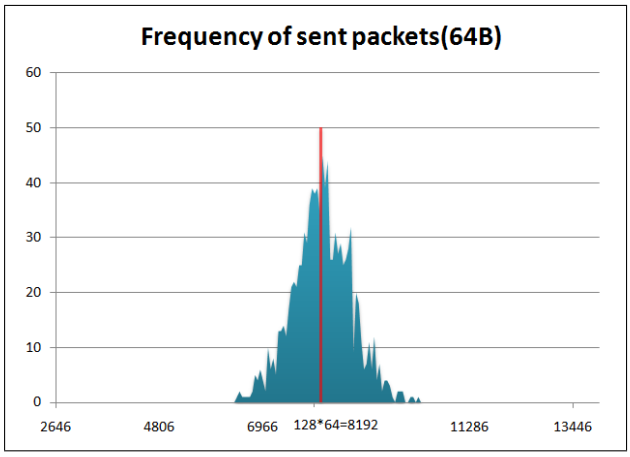


Figure 3: Distribution of the number of packets sended for one 64 bytes message attacked by Chopchop

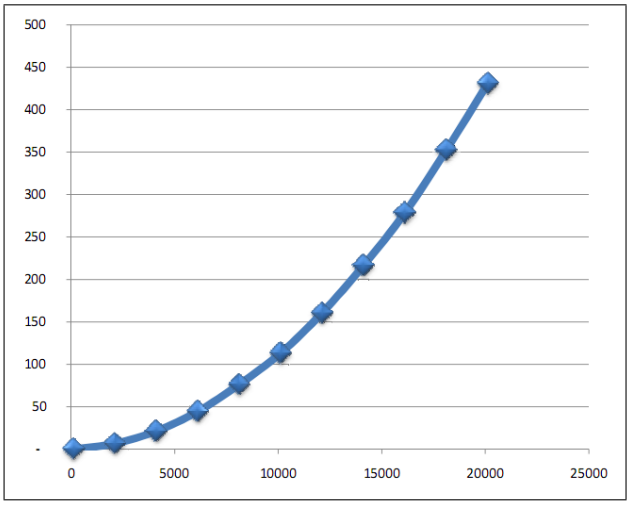


Figure 4: Chopchop time (s) vs length of one message (byte)

2.4 Application of the attack on a real network

This attack is useful because all known attacks against WEP need to see a lot of ARP packets. Thanks to this attack, it is not necessary to have an important traffic with a lot ARP requests. We just need to do a fake authentification which only requires to know the MAC address of a client. Then, we use a *keep alive packet* [2] to launch Chopchop and discover the keystream. We inject several ARP requests to have enough ARP responses. It generates a lot of traffic (see Figure 5). Finally, we use known attacks such as PTW to break RC4 and find the key.

We have launched the Chopchop attack with aircrak (see Appendix 1). We found one byte every 3 seconds in average. Then, we discovered a WEP key with PTW (see Appendix 2).

In addition to recovering the key, the attack enables to decrypt any packet at any time without the key.

[2] Any packet can be used but the fake authentification creates these packets at least

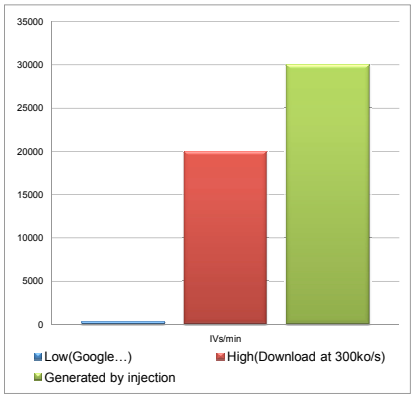


Figure 5: IVs obtained per minutes in different conditions

3 WPA, a more secure protocol?

3.1 Introduction to WPA

WPA was introduced in 2004 with several countermeasures. Because every device could not handle every point of the norm, different modes were created to match with existing hardware. Concerning authentification, there is two possibilites: the standard one with a RADIUS authentification server (quite expensive and difficult to set up) often called Enterprise Mode and a simple mode with a Pre-Shared Key (PSK). Concerning the data encryption there is also two possibilities: TKIP (Temporal Key Integrity Protocol) that uses key mixing and RC4 ciphering, or CCMP that is a encryption based on AES. We will describe the PSK-TKIP mode. It is used in personal and small firms networks.

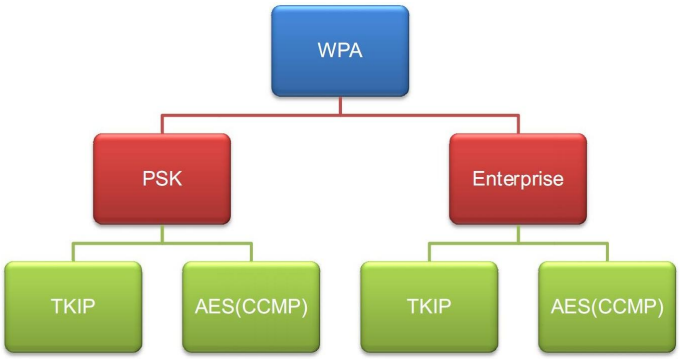


Figure 6: Different Modes of WPA

Authentification In opposition to WEP, there is an authentification to open one WPA session. At the end of the authentification, the client received keys, the PTK and the TMK which will be used during one session. It improves the security of a Wi-Fi. Indeed, every attack based on a Wi-fi sniffing or on a packet forgery only works during one session.

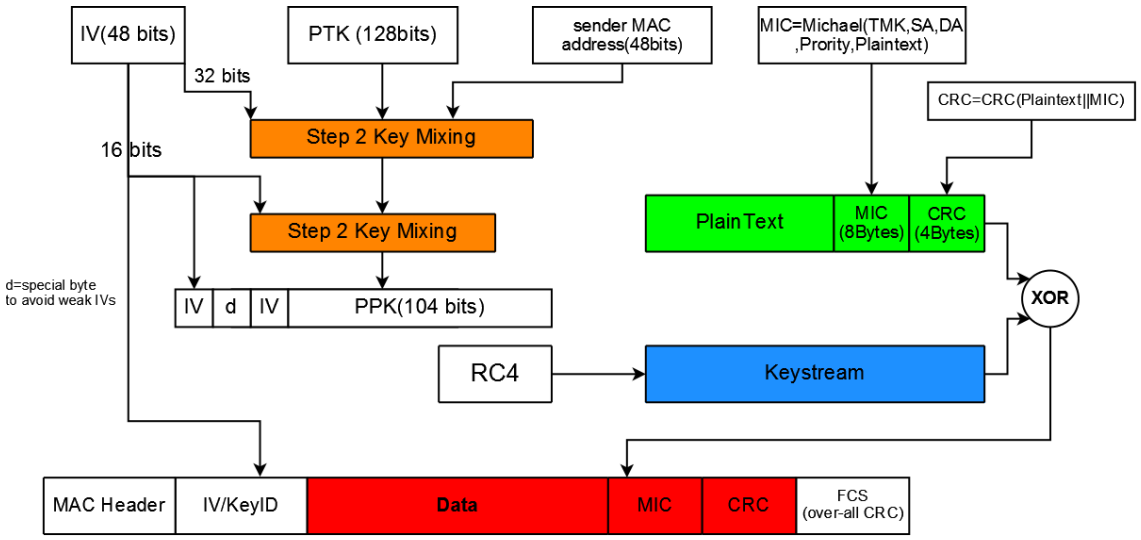


Figure 7: WPA Encryption

Key Mixing The key mixing builds the Per-Paquet-Key (PPK). It is composed of two phases. In the input of both phases, there is the packet IV, which includes the TSC, TKIP Sequence Counter. This TSC is incremented when a packet is received and valid. A packet with a too low TSC will be discarded by the AP or the client. Thus, the PPK will be different for every packet. As a result, all known attacks against RC4 cannot be used. They all require several packets with the same key.

MIC and CRC CRC32 is used as in the WEP. To improve the security, the Message Integrity Code, called Michael, was added. It enables to check if the sender and the receiver have been authentificated. It also check the priority channel. Indeed there are 8 Quality of Service channels which enables to route differently different kind of packets.

In opposition to the CRC, the MIC algorithm is not linear: it uses a Feistel function, B (see algorithm2), with a bits swap function, XSWAP (see algorithm3).

Although it is composed of 64 bits, it was designed to have 2^{20} possible values. In order to prevent a brute-force attack, a countermeasure was set up. If the AP or the client detects two corrupted MIC in one minute, a new authentification happens. A brute force attack in this way takes 2^{20} minutes.

However, MIC was designed to be computed by Wi-fi cards and APs[3]. Therefore, it requires few computations. Thus, defense against packets forgery is weakened. Indeed, MIC is not a one-way function and the computation of his inverse is as fast as the MIC (see Appendix 3).

Algorithm1: $Michael(K, (m_0, ..., m_{n-1}))$
Input: key K and message m
Output: MIC value (L,R)
$(L, R) = (K_{low}, K_{high})$
for $i = 0$ .. $n - 1$
do {
$L = L \oplus m_i$
$(L, R) = B(L, R)$
}
$return$ (L, R)

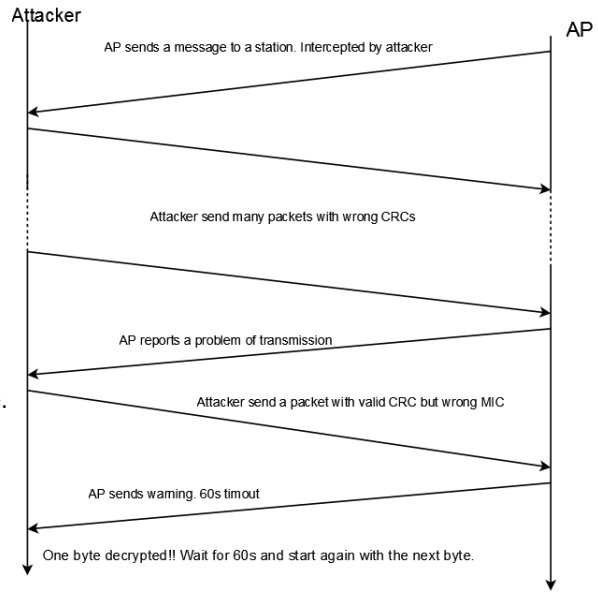
Algorithm2: $B(L, R)$
Input: (L,R)
Output: (L,R)
$R = R \oplus (L << 17)$
$L = L + R \ mod \ 2^{32}$
$R = R \oplus XSWAP(L)$
$L = L + R \ mod \ 2^{32}$
$R = R \oplus (L << 3)$
$L = L + R \ mod \ 2^{32}$
$R = R \oplus (L >> 2)$
$L = L + R \ mod \ 2^{32}$
$return(L, R)$

Algorithm3:
$XSWAP(ABCD)$
Input: 4 bytes A,B,C,D
Output: 4 swapped bytes
$return$ $BADC$

[3]The algorithm uses few cycles per bytes

3.2 Chopchop attack against WPA

In spite of all countermeasures, a chopchop-like attack can be set up against WPA.



While we send a packet with a false CRC, the AP consider the message has not been transmitted successfully. The MIC verification is not executed. Thus, we can run chopchop. We receive a ciphered message. When we guess the byte X and send a CRC valid message, the MIC verification fails and the AP sends a *MIC failure report frame*. We have to wait 60 seconds to guess the second byte of the ciphered message to avoid a new authentification. We can guess one byte of the keystream per minute.

However, when we send a packet, if the TSC is too low, the packet will be discarded by the AP. To avoid this, we have to run the attack on another Quality of Service channel. Indeed, every channel has his own TSC. A great part of the traffic is circulating on one channel. Therefore, the TSC of the ciphered message we have intercepted will be high enough. The QoS control field is in the MAC header which is not ciphered.

Moreover, to run this attack, we would have to increment the TSC counter to send each message. However, the TSC is checked only when a packet has been received with a correct CRC and a correct MIC. Therefore, we can do all the attack on one channel.

We now have one keystream. This keystream has been built with one TSC and the AP adress as the sender address. We inverse the Michael algorithm to obtain the MIC key. We will be able to build as many correct MIC values as we want to. We can send one packet on every QoS channel, that is to say 7 packets, to the client and only to it.

3.3 Results

We have implemented a simulation to run Chopchop against WPA given an ideal connection. We have checked the speed of one byte guessed by minute. Indeed, all operations are negliglible compared to the 60-seconds timeout. Even though sending and receiving packets can be slower in a real situation, it remains negligible. Indeed, in our previous results with aircrack on WEP, we decrypted one byte in 3 seconds in average (see Appendix 1). At the end, we also invert the Michael algorithm whose time is also small compared to one minute (see Appendix 3).

This countermeasure makes the efficiency of chopchop decreases. Indeed, in our previous results with WEP, we decrypted 15000 bytes in 250 seconds (see Figure 4). With WPA, we have deciphered only 4 bytes in this time.

3.4 Other weaknesses

The key mixing also has its weaknesses (see Figure 8). Given two Per-Packet-Keys with IVs with the same four last bytes, the key mixing can be inverted to have the PTK which will be used during one session. To find two corrects PPKs (104 bits) with good IVs, it takes 2^{105} operations. The inversion only takes 2^{38} operations.

With the PSK-TKIP mode, the initial key is a passphrase that is to say an ASCII chain. Therefore, generally, people use the most frequent caracter and even existing words. Consequently, WPA-passphrase are prone to dictionnary attacks. This kind of attack has already been implemented in aircrack software.

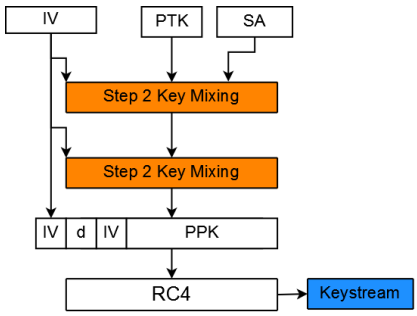


Figure 8: From PPK to the Keystream

4 Conclusion

We knew that WEP is totally broken. We can find the key easily and Chopchop enables to do it even when there is little traffic. Moreover, with Chopchop, we can decipher any packets without the key.

WPA is much more stronger. But the Chopchop attack is still possible. We can guess any keystreams and decipher any packets. However, it takes more time than with WEP. The attack has two additional drawbacks. On the one hand, the guessed keystream can be used only in the client direction. On the other hand, the keystream corresponds to one TSC, that is to say one Per-Paquet-Key which is not enough to break RC4. Thus, the main strength of TKIP is using different keys for each packet. It makes disappear all known attacks against RC4. Indeed, these attacks needs a lot of keystreams corresponding to the same key. Therefore, no attack on RC4 to find PPKs exists yet. Thus, the existing attack to go from the PPK to the PTK cannot be used.

However, sending 7 custom packets can be very dangerous: corrupted ARP packets can be send to do traffic hijacking.

Therefore, using the CCMP-AES mode instead of the PSK-TKIP mode enables to counter the chopchop attack. The CCMP-AES mode is imposed in WPA2. This protocol uses an AES encryption which is not prone to attacks.

References

[1] http://sid.rstack.org/blog/index.php/304-tkip-comment-ca-marche, Cedric Blancher, 7 November 2008

[2] M. Beck & E. Tews, Practical attacks against WEP and WPA, 8 November 2008

[3] http://sid.rstack.org/blog/index.php/305-des-fameuses-faiblesses-de-tkip, Cedric Blancher, 9 November 2008

[4] http://sid.rstack.org/blog/index.php/330-wpa-psktkip, Cedric Blancher, April 2009

[5] Paper IV: Weaknesses In The Temporal Key Hash of WPA, V. Moen & H, Raddum & K. J. Hole, 2006

[6] Security Analysis of Michael: The IEEE 802.11i Message Integrity Code & J. Huang & J. Seberry & W. Susilo & M. Bunder

[7] Analysis of Wi-fi Security Protocols and Authentification Delay, D. MKubulo, 2007

[8] IEEE Std 802.11-2007, 2007

[9] Sécurité Wi-fi, G. Pujolle, 2004

5 Appendices

5.1 Results with aircrack and Chopchop against WEP

```
aireplay-ng -4 -a  00:14:69:04:ED:C1 -h   00:21:5C:2D:ED:41  mon0
For information, no action required: Using gettimeofday() instead of /dev/rtc
The interface MAC (00:C0:CA:1A:07:25) doesn't match the specified MAC (-h).
        ifconfig mon0 hw ether 00:21:5C:2D:ED:41
Read 37 packets...

        Size: 116, FromDS: 1, ToDS: 0 (WEP)

             BSSID  =  00:14:69:04:ED:C1
          Dest. MAC  =  01:00:5E:00:00:05
         Source MAC  =  00:0E:39:AF:54:80

        0x0000:  0842 0000 0100 5e00 0005 0014 6904 edc1  .B....^.....i...
        0x0010:  000e 39af 5480 e0b6 7876 3800 6aed c7df  ..9.T...xv8.j...
        0x0020:  33f0 0a6c cd82 def7 f75f e00a 8b7e 88b3  3..l....._...~..
        0x0030:  67a2 98c9 46f9 4e12 a5ac 3a6a 2611 9d31  g...F.N...:j&..1
        0x0040:  7d13 8f60 a5f9 d56e 7b6f a3a1 d44d 0a19  }..`...n{o...M..
        0x0050:  2d04 f5d7 f4b2 2cb7 f8e3 a16c 71de 3e3f  -.....,....lq.>?
        0x0060:  d66e bfd0 9e0e 418e f5cf bbbf 2df6 75d1  .n....A.....-.u.
        0x0070:  ed9e 2ac5                                ..*.

Use this packet ? y

Saving chosen packet in replay_src-0617-070148.cap

Offset  115 ( 0% done) | xor = EC | pt = 29 |   86 frames written in  1857ms
Offset  114 ( 1% done) | xor = 8C | pt = A6 |   45 frames written in   958ms
Offset  113 ( 2% done) | xor = 51 | pt = CF |   64 frames written in  1372ms
Offset  112 ( 3% done) | xor = 1A | pt = F7 |  133 frames written in  2866ms
Offset  111 ( 4% done) | xor = D0 | pt = 01 |   68 frames written in  1449ms
Offset  110 ( 6% done) | xor = 75 | pt = 00 |  239 frames written in  5124ms
Offset  109 ( 7% done) | xor = F6 | pt = 00 |  178 frames written in  3853ms
Offset  108 ( 8% done) | xor = 2D | pt = 00 |   98 frames written in  2006ms
Offset  107 ( 9% done) | xor = BB | pt = 04 |   28 frames written in   569ms
Offset  106 (10% done) | xor = BB | pt = 00 |   14 frames written in   274ms
Offset  105 (12% done) | xor = CE | pt = 01 |  149 frames written in  3197ms
Offset  104 (13% done) | xor = F5 | pt = 00 |  256 frames written in  5510ms
Offset  103 (14% done) | xor = 8D | pt = 03 |  191 frames written in  4057ms
Offset  102 (15% done) | xor = 41 | pt = 00 |  172 frames written in  3647ms
Offset  101 (17% done) | xor = F8 | pt = F6 |  142 frames written in  3066ms
Offset  100 (18% done) | xor = 61 | pt = FF |  126 frames written in  2665ms
Offset   99 (19% done) | xor = D0 | pt = 00 |   37 frames written in   810ms
Offset   98 (20% done) | xor = BF | pt = 00 |  179 frames written in  3894ms
Offset   97 (21% done) | xor = 6E | pt = 00 |   51 frames written in  1137ms
Offset   96 (23% done) | xor = D6 | pt = 00 |  136 frames written in  2788ms
Offset   95 (24% done) | xor = 01 | pt = 3E |  219 frames written in  4715ms
Offset   94 (25% done) | xor = C0 | pt = FE |  185 frames written in  3950ms
Offset   93 (26% done) | xor = 86 | pt = 58 |   93 frames written in  1936ms
Offset   92 (28% done) | xor = F0 | pt = 81 |  480 frames written in 10233ms
Offset   91 (29% done) | xor = 44 | pt = 28 |   30 frames written in   622ms
Offset   90 (30% done) | xor = A1 | pt = 00 |   22 frames written in   450ms
Offset   89 (31% done) | xor = E3 | pt = 00 |   17 frames written in   381ms
Offset   88 (32% done) | xor = F8 | pt = 00 |  236 frames written in  5060ms
```

```
Offset   87 (34% done) | xor = B6 | pt = 01 |   30 frames written in   634ms
Offset   86 (35% done) | xor = 3E | pt = 12 |   74 frames written in  1588ms
Offset   85 (36% done) | xor = B8 | pt = 0A |   98 frames written in  2116ms
Offset   84 (37% done) | xor = F4 | pt = 00 |   39 frames written in   863ms
Offset   83 (39% done) | xor = 17 | pt = C0 |  128 frames written in  2804ms
Offset   82 (40% done) | xor = 0A | pt = FF |  185 frames written in  3998ms
Offset   81 (41% done) | xor = FB | pt = FF |   53 frames written in  1158ms
Offset   80 (42% done) | xor = D2 | pt = FF |  151 frames written in  3295ms
Offset   79 (43% done) | xor = 19 | pt = 00 |   66 frames written in  1449ms
Offset   78 (45% done) | xor = 0A | pt = 00 |   79 frames written in  1730ms
Offset   77 (46% done) | xor = 4D | pt = 00 |  174 frames written in  3698ms
Offset   76 (47% done) | xor = D4 | pt = 00 |  256 frames written in  5613ms
Offset   75 (48% done) | xor = A1 | pt = 00 |   62 frames written in  1330ms
Offset   74 (50% done) | xor = A3 | pt = 00 |   78 frames written in  1625ms
Offset   73 (51% done) | xor = 6F | pt = 00 |  255 frames written in  5539ms
Offset   72 (52% done) | xor = 7B | pt = 00 |  166 frames written in  3590ms
Offset   71 (53% done) | xor = 6E | pt = 00 |  173 frames written in  3705ms
Offset   70 (54% done) | xor = D5 | pt = 00 |  183 frames written in  3988ms
Offset   69 (56% done) | xor = 37 | pt = CE |   45 frames written in   987ms
Offset   68 (57% done) | xor = C2 | pt = 67 |    7 frames written in   147ms
Offset   67 (58% done) | xor = 60 | pt = 00 |   99 frames written in  2035ms
Offset   66 (59% done) | xor = 8F | pt = 00 |   57 frames written in  1224ms
Offset   65 (60% done) | xor = 4B | pt = 58 |  246 frames written in  5392ms
Offset   64 (62% done) | xor = FC | pt = 81 |   94 frames written in  2010ms
Offset   63 (63% done) | xor = F9 | pt = C8 |  240 frames written in  4997ms
Offset   62 (64% done) | xor = 9C | pt = 01 |  150 frames written in  3169ms
Offset   61 (65% done) | xor = 49 | pt = 58 |  214 frames written in  4495ms
Offset   60 (67% done) | xor = A7 | pt = 81 |  163 frames written in  3492ms
Offset   59 (68% done) | xor = 46 | pt = 2C |  212 frames written in  4556ms
Offset   58 (69% done) | xor = 3A | pt = 00 |  178 frames written in  3783ms
Offset   57 (70% done) | xor = AD | pt = 01 |   21 frames written in   434ms
Offset   56 (71% done) | xor = A7 | pt = 02 |   50 frames written in  1085ms
Offset   55 (73% done) | xor = 17 | pt = 05 |  216 frames written in  4700ms
Offset   54 (74% done) | xor = 4E | pt = 00 |  211 frames written in  4512ms
Offset   53 (75% done) | xor = F9 | pt = 00 |   91 frames written in  1973ms
Offset   52 (76% done) | xor = A6 | pt = E0 |  121 frames written in  2653ms
Offset   51 (78% done) | xor = F7 | pt = 3E |   16 frames written in   336ms
Offset   50 (79% done) | xor = 66 | pt = FE |   20 frames written in   422ms
Offset   49 (80% done) | xor = FA | pt = 58 |   74 frames written in  1548ms
Offset   48 (81% done) | xor = E6 | pt = 81 |   63 frames written in  1334ms
Offset   47 (82% done) | xor = 5E | pt = ED |   20 frames written in   438ms
Offset   46 (84% done) | xor = C6 | pt = 4E |   56 frames written in  1183ms
Offset   45 (85% done) | xor = 27 | pt = 59 |  138 frames written in  2903ms
Offset   44 (86% done) | xor = 8A | pt = 01 |   62 frames written in  1355ms
Offset   43 (87% done) | xor = 0A | pt = 00 |  190 frames written in  4090ms
Offset   42 (89% done) | xor = E0 | pt = 00 |  242 frames written in  5273ms
Offset   41 (90% done) | xor = 4F | pt = 10 |  199 frames written in  4295ms
Offset   40 (91% done) | xor = FD | pt = 0A |   21 frames written in   446ms
Offset   39 (92% done) | xor = BB | pt = 4C |  113 frames written in  2448ms
Offset   38 (93% done) | xor = DE | pt = 00 |  229 frames written in  4929ms
Offset   37 (95% done) | xor = 42 | pt = C0 |  233 frames written in  4974ms
Offset   36 (96% done) | xor = 88 | pt = 45 |   18 frames written in   377ms
Offset   35 (97% done) | xor = 6C | pt = 00 |  229 frames written in  4930ms
Offset   34 (98% done) | xor = 02 | pt = 08 |  147 frames written in  3073ms

Saving plaintext in replay_dec-0617-070532.cap
Saving keystream in replay_dec-0617-070532.xor
```

```
Completed in 218s (0.36 bytes/s)
```

5.2 Results with PTW after Chopchop

```
Starting PTW attack with 10284 ivs.

                                                                  Aircrack-ng 1.0 rc2 r1385

                                                        [00:00:56] Tested 793 keys (got 30682 IVs)

KB    depth   byte(vote)
0    0/  2   73(47872) 1E(40960) B0(39168) 43(37632) 6D(37632) 8F(37120) 08(36608) EA(36608) 13(3609
1    5/  1   51(36352) 20(35840) FD(35840) 10(35584) AD(35584) 39(35328) 4F(35328) 70(35328) F5(3532
2    0/  1   3D(44032) C7(36864) CB(36608) 06(36096) FB(35840) 90(35584) 9B(35584) DB(35584) F6(3558
3    6/ 18   94(36864) 04(36608) F3(36352) 37(35840) 77(35840) D3(35840) 3D(35584) 63(35584) 7E(3558
4    6/  4   7A(35840) 94(35584) A4(35584) E4(35584) E5(35584) EB(35584) 1F(35328) 78(35328) 8F(3532

     KEY FOUND! [ 73:64:66:67:68:6A:6B:6C:73:64:66:67:68 ] (ASCII: sdfghjklsdfgh )
        Decrypted correctly: 100%
```

5.3 Inverse of MIC algorithm

$Michael^{-1}(V, (m_0, ..., m_{n-1}))$
Input: mic value V and message m
Output: key (L,R)
$(L, R) = (V_{low}, V_{high})$
for $i = n - 1$ .. 0
do {
$(L, R) = B(L, R)$
$L = L \oplus m_i$
}
$return\ (L, R)$

$B^{-1}(L, R)$
Input: (L,R)
Output: (L,R)
$L = L - R\ mod\ 2^{32}$
$R = R \oplus (L >> 2)$
$L = L - R\ mod\ 2^{32}$
$R = R \oplus (L << 3)$
$L = L -\ mod\ 2^{32}$
$R = R \oplus XSWAP(L)$
$L = L - R\ mod\ 2^{32}$
$R = R \oplus (L << 17)$
$return(L, R)$