6 Chapter

How IEEE 802.11 WEP Works and Why It Doesn't

This chapter is dedicated to failure. It focuses entirely on WEP, the security method originally employed with Wi-Fi LANs and which has now been discredited due to its numerous security weaknesses. It may seem strange to devote so much space to a protocol that will soon be consigned to history. However, an understanding of WEP and its failure modes is very educational as a case study and highlights the areas that need to be addressed for real security. The first half of the chapter looks at the design of WEP and the second half shows why it fails to meet its security goals.

Introduction

For the first five years of its life, IEEE 802.11 had only one method defined for security. This was called **Wired Equivalent Privacy** or WEP (often misidentified as Wireless Effective Privacy and other variants). In 2000, as Wi-Fi LANs increased in popularity, they attracted the attention of the cryptographic community, who rapidly detected cracks in the WEP approach. By the end of 2001, tools were available on the Internet designed to crack open WEP in a fairly short time.

For many people, WEP is the only choice until the new security methods added to the IEEE 802.11 standard become established. Even with its weaknesses, WEP is still more effective than no security at all, providing you are aware of its potential weaknesses. It provides a barrier, albeit small, to attack and is therefore likely to cause many attackers to just drive on down the street in search of an unprotected network. Most of the attacks depend on collecting a reasonable sample of transmitted data so, for a home user, where the number of packets sent is quite small, WEP is still a fairly safe option. This section looks at how WEP works in detail, what its weaknesses are, and what an attacker has to do to break in.

Some people criticize the designers of the original IEEE 802.11 standard for creating WEP with inherent weaknesses. However, there are a few things that need to be taken into account. The first is that, at the time WEP was designed, it was not intended to provide military levels of security. As the name suggests, WEP was intended to make it difficult to break in—in the same sense that it is difficult to break into a building to connect to the wired LAN—but not impossible to break in. Section 8.2.2 of the 1999 IEEE 802.11 standard states the following as the objectives for WEP (quoted verbatim):

- It is reasonably strong: The security afforded by the algorithm relies on the difficulty of discovering the secret key through a brute-force attack. This in turn is related to the length of the secret key and the frequency of changing keys. WEP allows for the changing of the key (K) and frequent changing of the Initialization Vector (IV).
- It is self-synchronizing: WEP is self-synchronizing for each message. This property is critical for a data-link-level encryption algorithm, where "best effort" delivery is assumed and packet loss rates may be high.
- It is efficient: The WEP algorithm is efficient and may be implemented in either hardware or software.
- It may be exportable: Every effort has been made to design the WEP system operation so as to maximize the chances of approval, by the U.S. Department of Commerce, of export from the U.S. of products containing a WEP implementation. However, due to the legal and political climate toward cryptography at the time of publication, no guarantee can be made that any specific IEEE 802.11 implementations that use WEP will be exportable from the USA.
- It is optional: The implementation and use of WEP is an IEEE 802.11 option.

Notice that the requirements try to balance "reasonably strong" against the need for simple implementation and exportability. The issue of self-synchronization is really important for Wi-Fi LAN. Basically, what it says is that each packet must be separately encrypted so, given a packet and the key, you should have all the information you need. Clearly, you don't want a situation in which a single dropped packet makes all the following ones indecipherable.

The IEEE 802.11 standard only ever specified the use of 40-bit keys. As we have seen, 40 bits is too short to withstand serious brute force attack, which was why it was acceptable under export rules. The rationale was that if, say, a bank was intending to use wireless LAN, it would have its own security protocol running over the top of WEP and this security would be much higher, as appropriate to its application.

In retrospect, accepting this concept of a "reasonable" level of security was a mistake. Some people will argue that there are only two types of security: strong and none. The standard should probably have incorporated a really strong solution or taken a position that security had to be provided by some other means (like virtual private networking (VPN), for example). However, the power of marketing came to play and, in the promotion of IEEE 802.11 to the world, somehow the word "reasonably" was dropped in the brochures and WEP was simply described as secure. Furthermore, after export restrictions were relaxed, manufacturers made nonstandard extensions by using 104-bit keys. This step made them feel justified in adding adjectives like "extremely" and "absolutely" to the brochure. WEP was now *completely* secure, at least in the minds of the marketing managers. The long key extensions were adopted as part of the Wi-Fi specification and became the norm in the industry in 1999.

For the moment, let's step back from the marketing hype and look at how WEP works. To do that, we need to get back to the low-level IEEE 802.11 messages, some of which are covered in Chapter 5. All of the following refers to the 1999 standard. We cover the new security protocols in depth in a later section.

The IEEE 802.11 (1999) defined two levels of security: open and shared key. **Open security** really means *no* security. It is used in the same way that one would say, "I went to work and left the front door of my house open." Most people have figured out this is not a good security policy for their homes, and you probably feel the same way about Wi-Fi LANs. **Shared key** simply means that both ends of the wireless link know a key with a matching value. To be useful, this must be a secret shared only between trusted parties.

Authentication

There are two parts to WEP security described in the standard. The first is the authentication phase and the second is the encryption phase. The idea goes roughly as follows: When a new mobile device wants to join to an access point, it must first prove its identity. Ideally, the mobile device would also like the access point to prove itself as well. This phase is known as **authenticating** each other's identity. We need to delve into the concept of authenticating a bit more deeply here because authenticating in a WEP environment is a bit of a fool's errand.

The purpose of authentication is for each party to prove that he is who he claims to be. When you sign a check, you are authenticating yourself to the recipient, who will then use the signature to prove to the bank that you really wrote the check. In a LAN environment, every device has a (supposedly) unique number called the MAC address. Every transmission from a device on the LAN contains its MAC address so the identity of the sender can be checked. But how do you know that someone else didn't forge a message with a fake MAC address? One approach is to authenticate a device when it first joins the LAN and agree to a secret code that will be used to protect every subsequent message. Because only the true device and the access point know the secret code, each message can be validated as authentic when it is received. This is the purpose of authentication.

Now let's go back to IEEE 802.11 WEP. It has an authentication phase in which a new device proves that it is a trusted member of the group. We will look at how that is done in a moment. The access point reasons that, if the device can prove that it is trusted, it is reasonable to believe that the device's MAC address is true. Based on this trust, it will let the new device join. Unfortunately, however, in WEP no secret token is exchanged upon authentication. So there is no way to know whether the subsequent messages come from the trusted device or from an impostor. This authentication is really a rather embarrassingly pointless exercise and, in fact, was completely dropped from the Wi-Fi specification, despite being in the IEEE 802.11 standard.

As an analogy, imagine you hear a knock at the door and open it to find a man who has come to repair a utility fault inside your home. The man is wearing a utility company uniform and a mask with two holes for the eyes (okay, okay, bear with us for a moment). You ask for identification and he hands you a utility company badge. You even call up the utility company and confirm that he is scheduled to visit. The man comes in and then says he needs to go out to his van for a few minutes. In 30 seconds, a figure appears wearing the same uniform and mask and walks into your house. Question: How do you know it is the same guy? You don't know for sure. *So what was the point of checking in the first place if you can't positively identify the man every time he walks in?* You can now see the point of the mask in the analogy: In real life, we use our recognition of a person's face to confirm a person's authentication. But in a Wi-Fi LAN, there is no inherent way to do this. We will see later that the new security methods do provide this type of guarantee.

Despite its weakness, some systems still do use the "authentication" phase of the original IEE802.11 standard, so let's look at the messages that are exchanged. In the primer section, we point out that IEEE 802.11 uses three types of message: control, management, and data. The authentication phase uses management frames, as shown in Figure 6.1. For open authentication, the mobile device sends one message requesting authentication and the access point replies with a success message. For WEP-based authentication, an exchange of four messages occurs. First the

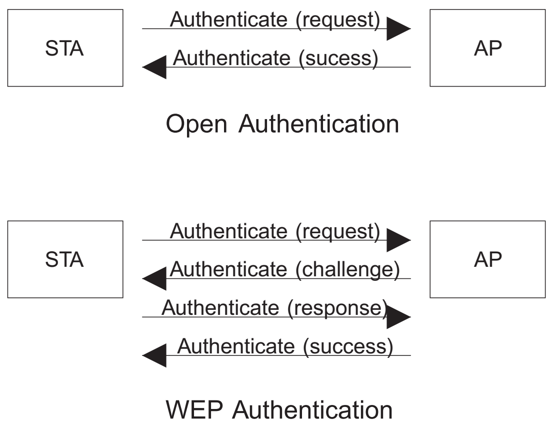

Figure 6.1 Authentication Sequences in the Original IEEE 802.11 Standard

mobile device requests authentication, and then the access point sends a challenge message. The mobile device responds to the challenge to prove that it knows a secret key and, if the proof is accepted, the access point sends the success message.

In principle, if the access point is operating in open mode, it always accepts the authentication request and responds with an authentication success message. This is the definition of open system operation. However, in practice many systems provide proprietary screening methods, the most popular being MAC address lists. The access point has a list of the MAC addresses that it will allow to join the network. This list is created by the manager and programmed in. The authentication is refused unless the mobile device's MAC address is found in the list. This doesn't protect against MAC address forgery, but it gives basic protection against very simple attacks using an off-the-shelf Wi-Fi LAN card, or even against accidental connection to the wrong network or another person's system.

WEP authentication is intended to prove to a legitimate access point that the mobile device knows the secret key. When the mobile device requests authentication, the access point sends a random number called **challenge text**. This is an arbitrary 128-bit number (preferably random). The mobile device then encrypts this number with the secret key using WEP and sends it back to the access point. Because the access point remembers the random number previously sent, it can check whether the result sent back was encrypted with the correct key; the mobile device must know the key in order to encrypt the random value successfully. Notice that this does nothing to prove to the mobile device that the access point knows the key. Notice also that if an attacker is listening, you just handed them a matching sample on which to start work because the challenge contains the plaintext and the response contains the ciphertext. You can start to see why the organization defining interoperability, the Wi-Fi Alliance, dropped the use of this exchange altogether.

The one benefit of the authentication exchange in a legitimate network is that it prevents stations joining the network unless they know the WEP key. There is a time savings in rejecting mobile devices that cannot communicate after associating. This is a management feature rather than a security feature. For example, if someone were to mistakenly enter the wrong key value, or fail to update his keys, the access point would reject the authentication and the user would be notified of the failure. Without the authentication phase, the mobile device is accepted, but every frame it sends is discarded by the access point due to decryption failure. From the mobile side, it is hard to distinguish this failure from failure due to interference or being out of range.

For completeness, let's look at the frame of the authentication messages used in this phase. Although multiple messages may be sent, they all have the same general format, as shown in Figure 6.2.

- The Algorithm Number indicates the type of authentication being used:
  0 – Open system
  1 – Shared key (WEP)
- The Transaction Sequence indicates where we are in the authentication sequence. The first message is 1, the second 2, and the third message (only used with WEP) is 3.
- The Status Code is sent in the final message to indicate success or failure of the authentication request.
- The Challenge Text field is used in the shared key (WEP) authentication, as described previously.

Privacy

If asked "What is the purpose of wireless LAN security," many people identify privacy as the key issue. It means preventing strangers from intercepting and understanding your data. Privacy is only one component of a security protocol and is not always needed, as the earlier analogy with signing a check shows. However, for Wi-Fi LAN security, privacy is a very desirable attribute, and was central to WEP's objectives.

Algorithm Num	Transaction Seq.	Status Code	Challenge Text

Figure 6.2 Authentication Message Format

Use of RC4 Algorithm

When WEP is enabled, the data messages are encrypted so an attacker who listens in cannot understand the contents (at least this is the intent). To decode the message, you have to know the secret key. The original IEEE 802.11 standard specified a two-phase approach: First you authenticate, and then you encrypt the data. As we have seen, the authentication method is next to useless; in fact, it is worse than useless because it gives an enemy information to use in an attack. Therefore, most Wi-Fi systems use open authentication and then switch on encryption after association. The fact that you effectively skip the authentication phase doesn't give the enemy any advantage in this case because, although he can join the network unchallenged, he can't send or receive any data without knowing the WEP keys for encryption.

The management of WEP keys is confusing because there are several of them used in different situations. We look at this in detail later; but for this section, let's assume that both the access point and the mobile device know the keys and that the mobile device has successfully associated (with or without authentication).

Security systems can be based around stream ciphers or block ciphers. A **stream cipher** takes a sequence of ordinary data (plaintext) and produces a sequence of encrypted data (ciphertext). It's like a sausage machine—you keep feeding plain bytes in one end and encrypted bytes come out the other end in a continuous process. A **block cipher** handles a single block of data at a time. It is more like a bakery—the dough is broken into lumps and each lump is processed separately to produce a loaf. In the case of data, fixed-length blocks are formed (typically 8, 16, or 32 bytes). Each block goes into the encryption algorithm and emerges as a completely different block of the same length. An important distinction between stream and block ciphers is that the internal state of a stream cipher is continuously updated as data is processed. By contrast, the state of a block cipher is reset for each block prior to processing.

WEP uses a stream cipher called RC4 to encrypt the data packets (Schneier, 1996). At the highest level, RC4 is a black box that takes in one byte from a stream at a time and produces a corresponding but different byte for the output stream, as shown in Figure 6.3. As with all encryption streams, the output stream is intended to look like a sequence of random numbers regardless of what the input stream looks like. Decryption is the reverse process and uses the same keys as for encryption (hence this is called a **symmetric algorithm**).

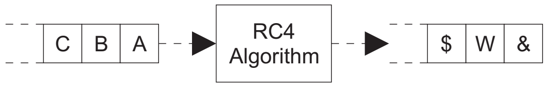

Figure 6.3 Stream Cipher

One of the advantages of RC4 is that it is fairly easy to implement and does not use any complicated or time-consuming operations like multiplication. Generally, this is the challenge for designers of an algorithm—to make it both secure and easy to implement. There are two main phases to RC4's use. In the first phase, initialization, some internal data tables are constructed based on the key value supplied; and in the second phase, the data runs through and is encrypted.

In the case of WEP, both the initialization phase and the encryption phase occur with each packet. That is, each packet is treated as if it were a new stream of data, which ensures that if one packet is lost the following packet can still be decrypted. This is both a strength and, as we shall see later, a source of weakness.

Initialization Vector (IV)

Before looking at the algorithm itself, we need to consider the encryption key again. As we mentioned previously, the original key length was 40 bits, which most manufacturers have increased to 104 bits. Manufacturers often refer to their 104-bit key solutions as "128-bit" security. So what happens to the extra 24 bits? The answer lies in the initialization vector.

There is a problem in using a fixed key value. Although the key may be updated from time to time, it is fixed relative to the flood of data packets running through the system. Effectively all the data packets are encrypted using the same key value. Suppose you initialize the RC4 algorithm with your key and run the message "qwertyuiop" into it. Suppose you get the encrypted result "b%fP*aF$!Y". This looks good and undecipherable. However, if the key is fixed, every time you run the same text "qwertyuiop" after initialization, you get the same result. In one sense, this is good—if you were to get a different result every time, it might make decryption somewhat tricky. But in another important way, this is very bad because it gives an attacker information. If she spots the same encrypted bytes in a given position, she knows that the original plaintext is being repeated.

How might this ability to spot repeated text be useful? Well, for example, the IP address always falls in the same place in a packet. So if you see the same encrypted bytes in that location, you know the message is from the same IP address (or going to the same IP address) as a previous message. You could think up lots of examples, but the basic principle is that you are giving information to the attacker and that is bad.

The solution to this problem is the **initialization vector (IV)**. This is a very simple concept. Instead of just using the fixed secret key to encrypt the packets, you combine the secret key with a 24-bit number that changes for every packet sent. This extra number is called the IV and effectively converts the 104-bit key into a 128-bit key. In our opinion, calling this 128-bit security is a minor con trick because the value of the IV is not secret at all but is transmitted openly with the encrypted frame.

To prevent the use of a fixed key for encryption, the actual key used to initialize the RC4 algorithm is the combination of the secret key and the IV, as shown in Figure 6.4.

Because the IV value always changes, the key used for encryption effectively changes with every packet so even if the input data (plaintext) is the same, the encrypted data (ciphertext) is always different.

The initialization vector is not a secret. In fact, it is sent openly as part of the transmission so the receiver knows which IV value to use in decryption. Any attacker can read the IV as well. In theory, knowledge of the IV is useless without knowledge of the secret part of the key. To be effective, the same IV value should *never be used twice* with a given secret key. Because the attacker can read the IV value, he could keep a log of the values used and notice when a value is used again. This would be the basis for an attack.

Unfortunately the IV in IEEE 802.11 WEP is only 24 bits long. This seems like quite a long number, but a few calculations show that it is not really enough. A 24-bit number has values from 0 to 16,777,216—so there are about 17 million IV values possible. A busy access point at 11Mbps is capable of transmitting/receiving about 700 average-sized packets a second. If a different IV value were used for every packet, all the values would be used up in less than seven hours! Because few people change their keys every day, IV values are bound to be reused.

There are other causes of IV reuse due to implementation issues. For example, many systems always start with the same IV value after a restart, and then the IV follows the same sequence as it is updated for each packet. Many systems change the IV according to a pseudorandom sequence—that is, a sequence that is superficially random but always follows the same sequence of numbers when started with a given value. If there are 20 mobile devices turned on in the morning, and they all start with the same IV value and follow the same sequence, then *the same IV value will appear 20 times for each value in the sequence.*

The problems with the IV illustrate that it is hard to design security protocols based on stream ciphers because the internal state of the encryption process is not

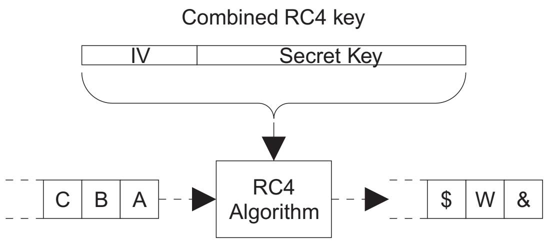

Figure 6.4 Using the IV

reset during a stream. Chapter 11 illustrates that the new version of WEP, called TKIP, which is also based on RC4, avoids IV reuse as an important part of the design.

WEP Keys

The way in which WEP keys are used is a great source of confusion to many people. Awkward terminology in the standard makes this situation even worse. The different types of keys in the standard have names that are confusing and misleading and, as a result, many manufacturers have tried to "help" by inventing new terms that are more easily understood. This is a double-edged sword because, while the new terms are better, they are not consistent across manufacturers, with the result that there are now multiple names in use for the same concepts. No wonder people have difficulty in understanding WEP keys.

So as not to propagate this confusion, **we only use the terms used in the original standard**, so you need to do some homework: Learn the official terms and, if required, cross reference them to the names used in the system you have installed. There are only two types of WEP keys mentioned in the standard. The correct terms are:

- Default key(s)
- Key mapping key(s)

Table 6.1 is a translation table showing the user-friendly names manufacturers use for these terms.

WEP keys have the following characteristics:

- **Fixed length:** Usually 40 bits or 104 bits
- **Static:** No changes in key value except by reconfiguration

Table 6.1 Manufacturer Names for WEP keys

Standard Term	Manufacturer's Term
Default key	Shared key Group key Multicast key Broadcast key Key
Key mapping key	Individual key Per-station key Unique key

- **Shared:** Access point and mobile device both have copy of the same key(s)
- **Symmetric:** Same key used to encrypt and decrypt information

The keys are static, and both the mobile device and the access point must have a copy. So the question arises how to configure the key into both devices in a way that does not risk the key being discovered. The IEEE 802.11 standard conveniently bypasses this problem with the words *"The required secret is presumed to have been delivered to participating STAs via a secure channel that is independent of IEEE 802.11."* This is not an unreasonable position because the method of installing the keys is bound to vary for different types of devices. You can install the keys on a laptop computer, for example, by typing them or using a floppy disk. However, if you have a mobile phone with Wi-Fi LAN built in, you need to use a different approach—for example, using a smart card.

Although skipping the issue of key distribution is reasonable for the standard, it doesn't make the job any easier for the system manager. At home, it is relatively easy. Just bring up a configuration utility provided with the system, choose the key values, and type them in. You can easily manage a few tens of users in this way. However, in a corporation with dozens or even hundreds of users, installing the keys to all the mobile devices and access points is an absolute nightmare—especially because it is essential to avoid unscrupulous people finding out the key values. Changing the keys can also be a major undertaking, so it doesn't help when the security guru tells you to change keys every seven hours to avoid IV reuse! In practice, each vendor tends to use a different approach to solve this problem. Most allow the simple option of typing in the key value using a utility or via the client driver interface. Some allow keys to be distributed to the mobile stations on a floppy disk that contains the keys in some obscured format. Sophisticated methods for secure key distribution have been invented, but these are not included in the WEP specification. If you are using WEP, you'll probably have to put up with typing the keys in for now.[1]

There are two different approaches to using keys under WEP. Usually when there are two ways to do something in a standard, you can figure there was a technical argument in the standards committee that neither side could win. In the end, they included both approaches and assumed the market would decide which to use. In fact, in the case of WEP, both approaches are useful in different circumstances.

- In the first case, all the mobile devices and the access points use a single set of keys. These keys are called **default keys.**

1. A few vendors have attempted to provide key distribution with WEP, notably Cisco LEAP (see Chapter 9).

- In the second case, each mobile device has a key that is unique. In other words the key used between each mobile device and the access point is specific to that connection and not known to other mobile devices. These keys are called **key mapping keys.**

The two modes are shown diagrammatically in Figure 6.5. Note that in the first case all the devices need to know only one key between them. In the second case, the mobile devices each know one key, but the access point has to have a table of all the keys. You can see that the names "default" and "key mapping" used in the standard don't relate very well to the function, which is why manufacturers have coined new terms in their documentation. Just remember default keys are shared, and mapping keys map to a specific device.

Default Keys

Consider a scenario in which you want all the mobile devices to share the same secret key, as in a home installation and in most small- to medium-sized commercial organizations. You decide to use the default keys. In fact, many manufacturers only support default key operation so this may be your only option. When you go to enter your key, you are surprised to find that there is not one, but four, default keys in the system. The IEEE 802.11 standard specifies that there should be four

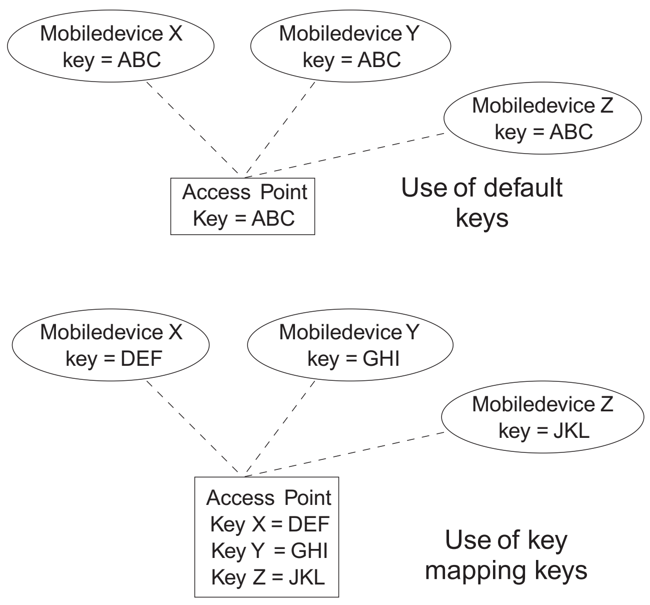

Figure 6.5 Difference Between Default and Key Mapping Keys

default keys for each device. This is another great source of confusion—do you need to enter all four? Do you need the same four on every mobile device as well? The level of confusion is so high that some users get as far as this configuration screen and give up, opting to take a chance without using WEP.

As with many things, these options make sense when you understand why you have them. Here are two key facts that help:

1. Only one default key is needed for security to work.
2. Multiple default keys are supported to help you change keys smoothly.

Let's suppose there was only one default key available system-wide. When you install your Wi-Fi LAN system, you choose a key and program it into the access point. You program the same key into each of the mobile devices and into new ones that you add later. Everything is fine until, like a good security manager, you decide it is time to change the key value. How do you do it? If you change the key in the access point, all the mobile devices will be disconnected immediately. Then you will have to track down all the users and reprogram their mobile device with the new key. This step could take hours, or days if people are out of the office. You could send out a memo telling everyone that from 9:00 Monday the key is changing and they need to update their computers. However, at least half the people will forget and be on the phone complaining their Wi-Fi LAN connection is broken and, in any case, you can hardly publish the key value in the memo for them to type in. If there were only one default key, changing its value would be a real pain in the neck.

The answer is to use two default keys "simultaneously." It works like this: When there are two default keys defined, all transmissions are encrypted using a single key that you select. This is called the **active key**. However, received frames can be decrypted using either of the two keys as appropriate. In summary, if you have multiple default keys you always encrypt using the active key but you can decrypt using any default key that is appropriate.

This technique of multiple keys makes key change much easier. The key change works as shown in Figure 6.6 (A–D).

In Figure 6.6A, the mobile station and access point are communicating using the first default key ABCDEF. The second default key is not assigned (this is called a null key). Now the manager decides to change the key. The first thing to do is to program the new key JKLMNP into to the access point as the second default key, as shown in Figure 6.6B. Note that the access point still transmits using ABCDEF—that is still the active key. By the way, you should never use such weak key values as these in a real system!

The next stage in the change is to notify all the users that they need to have their keys updated. Perhaps you go to their desks or ask them to come to you. In

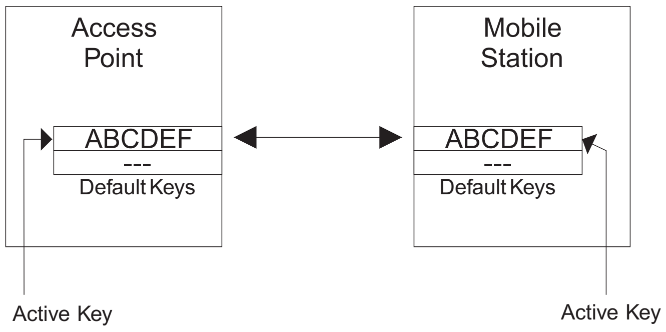

Figure 6.6A Before Changing Keys

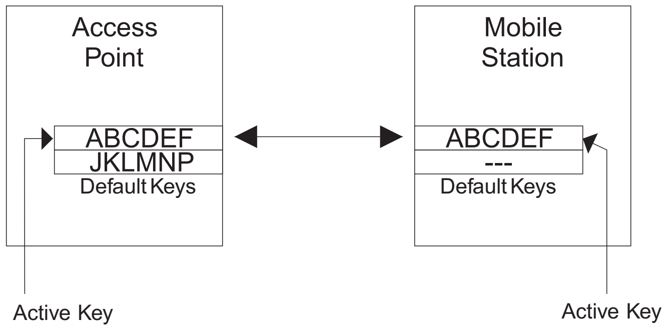

Figure 6.6B Adding a Second Default Key

each case you install the new secret key and *make the new key active on their devices*. Now the users whose devices have been updated will be transmitting using the new key, as shown in Figure 6.6C.

Notice that the access point is still sending using the old key, but the active key on the mobile has been changed. The access point must use the old key because some users have not been updated yet and don't know the new key. Access point transmissions using the old key are accepted by mobile devices that have not been updated and also by the ones that have the new key. The mobile devices with the new key still have a copy of the old key available. In the same way the access point can accept messages from both updated and un-updated mobile devices because it also has both keys available.

After all the users' devices have been updated, or after a cut-off date for the change, you move the access point over to the new key and the old key is deleted, as shown in Figure 6.6D. You can see how this key change has been possible with relatively little disruption because of the availability of multiple default keys. You

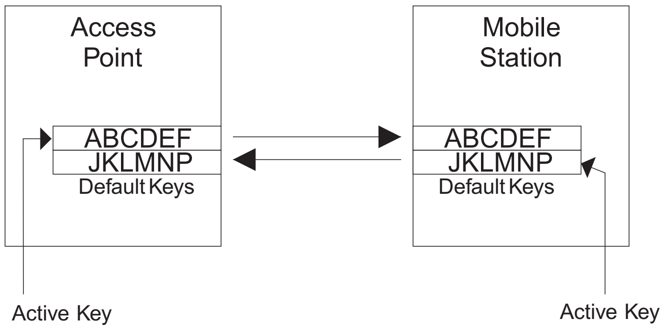

Figure 6.6C Use of Both Old and New Keys

need only two default keys to make this transfer work, so why are there four default keys? Was the Standards committee feeling generous on that day or is there some other reason?

We have assumed so far that the keys are used *bidirectionally*. In other words, the same key is used to receive and to transmit (except during a key change). With four keys, you can operate with different keys in each direction. Remember that transmitted frames are always encrypted using the active key. The active key is identified using a number: 0, 1, 2, or 3. So, for example, the AP might transmit using default key 0. However, there is no reason why the mobile devices have to transmit using 0. The mobile devices might all be configured with an active key of 2. Think about what that means—the AP encrypts its messages using key 0, but the mobile devices encrypt using key 2. Key 2 on the AP has to match key 2 on the mobile device and key 0 on the mobile has to match key 0 on the AP. However, key 0 is different from key 2. This is called **directional key use**.

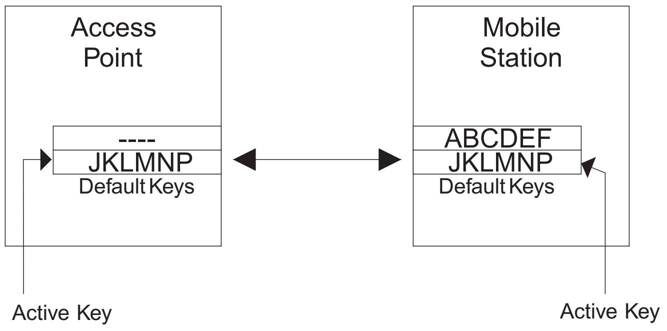

Figure 6.6D Completed Key Update

If you think directional key use gives more security, you could decide to always use keys 0 and 1 for AP transmissions and keys 2 and 3 for mobile device transmissions. You need two keys in each case to allow the key change to occur for each direction separately, making a total of four keys in use.

The key number (0, 1, 2, or 3) that is used for transmission has to be notified to the receiver so it knows which key to use for decryption. This is done by sending the information, called the **KeyID bits**, in each encrypted frame.

Now you know why there are four keys available. Remember that you are not obliged to use all four. You can operate quite happily with only one default key, provided you don't mind disruption when changing the value. There is one more use for default keys, which is needed when you use the key mapping keys, as described in the next section.

Key Mapping Keys

The basic principle of key mapping keys is to give each mobile device its own key value. The benefits of this approach are obvious in a large organization. If you have 1,000 users on a site and they all share a single default key, you will have a very hard time keeping that key secret and performing key updates. If instead each user has a unique key, you can change an individual key and enable or disable single users if, for example, a laptop is lost or stolen.

Not all manufacturers support key mapping keys because of the complexity of configuring and maintaining them. If you want to use this feature, you may need to shop around (and if you are shopping around now, we suggest you skip WEP and go straight to the new security approach of Wi-Fi WPA). Anyway, we briefly run through the issues here for completeness.

Using different keys per mobile device introduces a small but important complication related to broadcast handling. The architecture of most LANs derives from the idea of *sharing* a communication channel. For example, early wired LANs used a coax cable, rather like the antenna cable on a TV, to connect all the computers in series. The cable was shared between all the computers. Even today with modern Ethernet cabling, LAN hubs usually connect all the ports together so that they act like a single wired connection. In the wireless environment, radio transmissions are obviously shared due to the fact that anyone can receive them. In a shared LAN, three types of message operate: **unicast messages**, which are sent to a single destination, **group messages**, which are sent to several destinations at once and **broadcast messages**, which are sent to everyone. The last two cases are collectively called multicast messages.

Here's the rub: If every mobile device uses a different key, how does the AP send a broadcast message? Which key should it use for encryption? The solution is that all multicast traffic is sent using a default key that is shared by all the mobile devices. Only unicast messages are sent using the mapping key that is specific to the receiver. You need to program a minimum of two keys into each mobile device.

Apart from the need for two keys, implementing key mapping keys at the mobile device end is the same as for default keys. You must, for example, put the special key for this device into key number 0. The broadcast key goes into one of the other locations, say key number 2. You would set key 0 as the active key; mobile devices do not sent broadcasts, they only receive them.

The access point is much more complicated because it needs to know the special key for every possible station that might want to associate. Potentially this means keeping a table of hundreds of entries. Whenever the AP receives a frame, it has to pick out the sender's MAC address and use it as an index into the table to find the right key to use for decrypting. Similarly, before transmitting, it has to look up the correct key based on the destination address. This is tough work for the AP and you can see why some manufacturers opted not to provide support. The other problem is that the list of keys takes up memory space and, if there are several APs, they all have to have the same copy of the list. This makes management in a large system difficult and error prone.

The key mapping key option is a good idea, but it is not supported by all AP vendors and therefore has not been widely deployed. If you are interested in this type of approach, you will find that the new security methods such as WPA do provide these capabilities—and the ability to manage the key lists in a much more effective way.

By the way, the AP can operate with default keys and key mapping keys simultaneously; you are not required to switch between one or the other modes. When the AP receives a frame (or wants to send one), it looks in the key table to see whether there is an entry corresponding to the MAC address of the mobile. If it finds an entry, it uses it. If not, it uses the default key instead. *Eureka! That is why it is called the default key.* And that is why almost no one uses the term "default key" outside the standard (and this book).

Mechanics of WEP

So far we have discussed the original goals of WEP and given an overview of the design and use of keys. Now we will look in much more detail at the way in which WEP is implemented. You need to understand where the weaknesses of WEP arise.

Fragmentation

We are used to using the telephone. There was a time when all telephones looked the same. This was partly due to the monopoly of the telephone companies but probably was also due to the need for people who had not grown up with telephones to feel comfortable with the user interface they had learned. Today, telephones are part of our psyche. We can handle phones in any shape or size as naturally as catching a ball or walking on unfamiliar ground. We pick up the phone, dial the number, and talk to the other person with little effort. We don't care about the many complicated processes that occur between our lips and the ear of the receiver (that sounds a bit weird. but we're talking about electronic processes). A similar situation has evolved for application programs. There was a time when a programmer needed to know about the details of the network protocols and hardware that were used for communication. Today, the operating systems environment allows application programs to make a connection and communicate data with ease and without knowledge of the network implementation.

If a network includes a Wi-Fi LAN link, data from the operating system or a driver needs to pass to the IEEE 802.11 MAC service layer. In other words, a packet of data arrives at the Wi-Fi LAN with instructions to send it out. This packet of data is called an MSDU (MAC service data unit). If things go well, this MSDU will eventually pop out of the MAC service layer on the destination device and be passed to the operating system or driver for delivery to the target application. Before it reaches the radio for transmission, however, the MSDU may be broken up into several smaller pieces, a process called **fragmentation**. Each fragment is processed for WEP encryption. A MAC header is added to the front and a checkword added to the end.

You can see that the original MSDU may now be spread across several smaller messages and have had more bytes added on—quite apart from the fact that it will now be encrypted. Each one of the smaller messages is called an MPDU (MAC protocol data unit). We'll look at the last few stages, during which an MPDU encounters the encryption process.

The process treats the data as a block of unformatted bytes; the size depends on the original MSDU contents and the fragmentation settings. It is typically in the range of 10–1,500 bytes. The first step in encryption is to add some bytes called the **integrity check value (ICV)**.

Integrity Check Value (ICV)

The idea behind the ICV is to prevent anyone from tampering with the message in transit. In both encrypted and unencrypted messages, a check is made to detect whether any bits have been corrupted during transmission. All the bytes in the message are combined in a result called the CRC (cyclic redundancy check). This 4-byte value is added on to the end of the frame immediately prior to processing for transmission. Even if a single bit in the message is corrupted, the receiving device will notice that the CRC value does not match and reject the message.

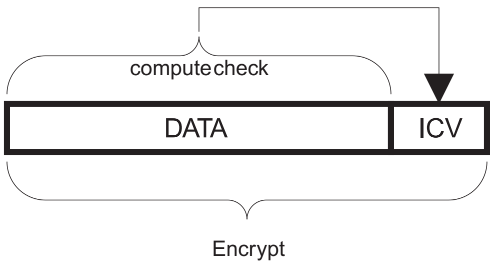

Figure 6.7 Adding the ICV

While this detects accidental errors, it provides no protection from intentional errors because an attacker can simply recompute the CRC value after altering the message and ensure that it matches again.

ICV is similar to the CRC except that it is computed and added on *before* encryption. The conventional CRC is still added after encryption. The theory is that, because the ICV is encrypted, no attacker can recompute it when attempting to modify the message. Therefore, the message is rendered tamper-proof. Er… well, only in theory, as we'll see shortly that clever people found a loophole. For now, let's suppose it works as intended.

So the ICV is computed by combining all the data bytes to create a four-byte checkword. This is then added on the end, as shown in Figure 6.7.

Preparing the Frame for Transmission

After the ICV is appended, the frame is ready for encryption. First, the system must select an IV value and append it to the secret WEP key (as indicated by the active key selection). Next, it initializes the RC4 encryption engine. Finally it passes each byte from the combined data and ICV block into the encryption engine. For each byte going in, an encrypted byte comes out until all the bytes are processed. This is a **stream cipher**.

For the receiver to know how to decrypt the message, the key number and IV value must be put on the front of the message. Four bytes are added for this purpose. The first three bytes contain the 24-bit IV value and the last byte contains the KeyID number 0, 1, 2, or 3, as shown in Figure 6.8.

Finally, the MAC header is attached and the CRC value placed at the end to detect transmission errors. A bit in the MAC header indicates to the receiver that the frame is WEP encrypted so that it knows how to handle it.

The receive process follows logically. The receiver notes that the WEP bit is set and therefore reads and stores the IV value. It then reads the key ID bits so it can select the correct WEP key, append the IV value, and initialize the RC4 encryption

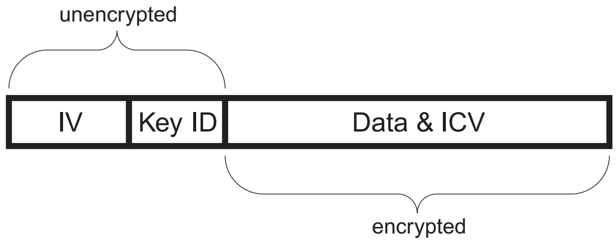

Figure 6.8 Adding the IV and KeyID bits

engine. Notice that with RC4 there is no difference between the encryption and decryption processes. If you run the data through the encryption process twice, you get back to the original data—in other words, the second encryption cancels out the first. Therefore, you need only one engine for both encryption and decryption. After the encryption engine is initialized, the data is run through one byte at a time to reveal the original message. The final step is to compute the ICV and verify that the value matches that in the received message. If all is well, the data portion is passed up for further processing.

RC4 Encryption Algorithm

RC4 is the name of the encryption algorithm used by WEP. An encryption algorithm is just a set of operations that you apply to plaintext to generate ciphertext. Obviously it is not helpful unless there is a corresponding decryption algorithm. In the case of RC4, the same algorithm is used for encryption and decryption. The value of an encryption algorithm lies partly in how strong it is and partly in how easy it is to implement. The strength of an algorithm is measured by how hard it is to crack the ciphertext. There certainly are stronger methods than RC4. However, RC4 is remarkably simple to implement and considered to be very strong if *used in the right way.* This last point is important because we will see all the weakness of WEP later, and those weaknesses do not derive from faults in RC4, but in the way it is applied in the case of WEP.

RC4[2] is a proprietary stream cipher designed in 1987. While the algorithm has received a great deal of public attention, RSA Labs, Inc. regards the description of the algorithm as a *trade secret. Implementers should consult RSA Labs about this issue. However, the algorithm was reverse-engineered and made public anonymously in 1994.*

Fortunately, because RC4 is simple to implement, it is also simple to describe. The basic idea behind RC4 encryption is to generate a pseudorandom sequence

2. RC4 stands for the fourth cipher designed by Ron Rivest (Rivest Cipher 4).

of bytes called the **key stream** that is then combined with the data using an exclusive OR (XOR) operation. For those not familiar with the XOR operation, it combines two bytes and generates a single byte. It does this by taking and comparing corresponding bits in each byte. If they are equal, the result is 0; if they differ, the result is 1. An example is shown in Figure 6.9.

XOR is often written mathematically using the symbol "⊕" so the example shown in the figure would be written:

```
00110101 ⊕ 11100011 = 11010110
```

One important characteristic of XOR is that if you apply the same value twice, the original value is returned:

```
00110101 ⊕ 11100011 = 11010110
11010110 ⊕ 11100011 = 00110101
```

In other words, if $A \oplus B = C$, then $C \oplus B = A$. You might guess that in the case of RC4, this property is exploited as follows:

```
Encryption: Plaintext ⊕ Random = Ciphertext
Decryption: Ciphertext ⊕ Random = Plaintext
```

It is necessary that "random" looks random to an attacker but that both ends of the link can generate the same "random" value for each byte processed. It is therefore called **pseudorandom** and is generated by the RC4 algorithm.

The most important property of a pseudorandom key stream is that you can calculate the next byte in the sequence *only* if you know the key used to generate the stream. If you don't know the key, it really looks random. Note that the XOR operation completely hides the plaintext values. Even if the plaintext is just a long series of 0 values, the ciphertext still looks random to an attacker.

XOR is a trivially easy operation for a computer to implement so the only challenge is to generate a good pseudorandom number stream. You need one

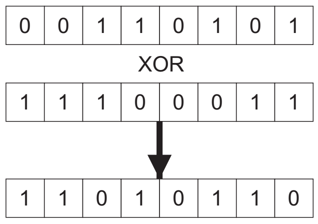

Figure 6.9 XOR Operation

pseudorandom byte for each byte of the message to be encrypted. RC4 generates such a stream.

There are two phases in RC4: key setup and pseudo-random generation. The first phase, the key setup algorithm, establishes a 256-byte array with a permutation of the numbers 0–255; that is, all the numbers are present in the array but the order is mixed up. The permutation in the array, or S-box, is established by first initializing the array with the numbers 0–255 in order. The elements in the S-box are then rearranged through the following process. First, a second 256-byte array, or K-box, is filled with the key, repeating as needed to fill the array. Now each byte in the S-box is swapped with another byte in the S-box. Starting at the first byte, the following computation is made:

```
j = (Value in first byte of S-box ) + (Value in first byte of K-box)
j is a single byte value and any overflow in the addition is ignored.
```

Now *j* is used as an index into the S-box and the value at that location is swapped with the value in the first location.

This procedure is repeated another 255 times until each byte in the S-box has been swapped. The process is described by the following pseudocode for people familiar with programming:

```
$i = j = 0$;
For $i = 0$ to 255 do
  $j = (j + S_i + K_i)$ mod 256;
Swap $S_i$ and $S_j$;
End;
```

Once the S-box has been initialized, the next phase in RC4 is the pseudorandom generation phase. This phase involves more swapping of bytes in the S-box and generates one pseudorandom byte per iteration (R). The equations for the iterations are shown here.

```
$i = (i + 1)$ mod 256
$j = (j + S_i)$ mod 256
Swap $S_i$ and $S_j$
$k = (S_i + S_j)$ mod 256
$R = S_k$
```

To generate the ciphertext, each byte of plaintext is XORed with a value of R as produced by the RC4 algorithm. Notice how the whole process has been done using byte length additions and swaps—very easy operations for computer logic.

Theoretically, RC4 is not a completely secure encryption system because it generates a pseudorandom key stream, not truly random bytes. But it's certainly sufficiently secure for our application, if applied correctly to the protocol.

Using XOR in this way is similar to a Vernam cipher (Vernam, 1926). Gilvert Vernam developed this cipher during World War I while working for AT&T. However, it is only completely secure if *R* is a *true* random byte (Menenez et al., 1996). What constitutes a *true random byte* is a philosophical debate we won't open here. But suppose that you generated a huge table of random numbers by sampling cosmic noise or some genuinely random physical event rather than using a pseudorandom algorithm such as in RC4. You could use your random numbers by sending a secret copy to your friend for use with the Vernam cipher. This has the advantage that the numbers are truly random *and* known to both ends of the link. This approach is very secure, but you would be allowed to use the table only once. Then you would have to eat it or otherwise dispose of it—it is a **one-time pad**. Ensuring that a one-time pad system is completely secure requires never reusing the same series of random bytes twice. Because the former Soviet Union made this serious mistake following World War II, the American National Security Agency (NSA) was able to decrypt a number of enciphered messages sent by Soviet agents in a project code named VENONA (U.S. NSA, 1999).

Why WEP Is Not Secure

WEP was included in the original IEEE 802.11 standard in 1997, but it was not until 1999 that systems were widely deployed when IEEE 802.11b and Wi-Fi became established. Most vendors added key extensions allowing a 104-bit key to be used (128 bit if you count the IV), and this was also adopted by Wi-Fi.

The industry started to have concerns about WEP security as designs were made and engineers started to point out some problems. In particular, the weakness of the authentication method was noted, as described earlier, and the authentication phase was dropped altogether. However, the manufacturers' concerns related to the strength of the security rather than the overall integrity. In other words, they were concerned that a serious and major attack might succeed. Nobody thought that it would be easy to break WEP.

Because of these concerns and also because of the difficulties in key management discussed in this chapter, the IEEE 802.11 Standards Committee launched a new task group to look at upgrading the security approach. The IEEE 802.11 committee is, essentially, a voluntary organization. Meetings are held roughly six times a year and the meetings are open to any industry members. The process involves technical presentations, discussions, drafting text, and a lot of voting. It is a democratic and parliamentary process. As a result, new standards do not develop quickly. Major modifications such as the security subsystem take years rather than months.

While the IEEE 802.11 group was considering a new approach, the security research community was also looking at WEP. In the early days, security gurus had not paid much attention because wireless LAN was not a widespread technology. But by 2000 it was appearing everywhere and many universities had installations. Security researchers had direct contact with Wi-Fi LAN. As did most people, they realized the potential advantages; but, unlike most people, they immediately questioned whether the security provisions were bomb-proof.

The answer came in 2000 as a series of reports emerged highlighting weakness after weakness (Walker, 2000; Arbaugh et al., 2001; Borisov et al., 2001). Finally, an attack was published showing that the keys could be extracted in a busy network within hours, regardless of the key length. The press were ecstatic. Security weakness and security breaches sell copy, and headlines appeared around the world. Even the main public media channels covered the event. This was a measure of how far IEEE 802.11 had come that it was considered of interest to the general public. However, there were lots of red faces—embarrassed manufacturers and angry customers.

This section looks in detail at the attacks on WEP that have been identified. WEP is a general term that covers a number of security mechanisms. As a basis for evaluating WEP, let's review the mechanisms that are needed for security:

- Authentication
- Access control
- Replay prevention
- Message modification detection
- Message privacy
- Key protection

Unfortunately, WEP fails to perform in all these areas. We'll look at each one separately.

Authentication

Authentication is about one party proving to the other that he really is who he claims to be. Authentication is not a one-time process—in other words, it is not enough to prove once that you are authentic. It is only useful if you can prove it every time you communicate. The process of authentication is often time consuming, and so a common approach is to perform full authentication on first contact and then provide a limited-life "identity badge." Ideally, the identity badge is such that it cannot be transferred to someone else. A photo ID is an example in which the government or corporation authenticates you.

In the wireless world, you usually need *mutual* authentication. The network wants proof about the user, but the user also wants proof that the network really is the expected one. This is important for Wi-Fi LANs because it is so inexpensive to set up decoy access points.

Finally, security experts point out that it is essential to use different secret keys for authentication than you use for encryption.[3] The use of derived keys is recommended because master keys should rarely or never be exposed directly to attack. In summary, the basic requirements for authentication in wireless LANs are:

1. Robust method of proving identity that cannot be spoofed
2. Method of preserving identity over subsequent transactions that cannot be transferred
3. Mutual authentication
4. Independent keys independent from encryption (and other) keys

Unfortunately, WEP fails on all counts. As a reminder, WEP authentication relies on a challenge–response mechanism. First, the AP sends a random string of numbers. Second, the mobile device encrypts the string and sends it back. Third, the AP decrypts the string and compares to the original string. It can then choose to accept the device and send a success message.

The key used for this process is the same WEP key used for encryption, thus breaking rule 4. The operation does not authenticate the access point to the mobile device because a rogue access point can pretend it was able to check the encrypted string and send a success message without ever knowing the key. Hence rule 3 is broken.

Rule 2 is broken because there is no token provided to validate subsequent transactions, making the whole authentication process rather futile.

Rule 1 is rather irrelevant given the weaknesses already pointed out, but it's quite interesting to look at why this also fails.

During authentication the access point sends a random string of 128 bytes. The way in which this "random" string is generated is not defined, but one would hope at least that it was different for each authentication attempt. The mobile station encrypts the string and sends it back. Sounds good, but hang on a moment—WEP encryption involves generating a sequence of pseudorandom bytes called the key stream and XORing it with the plaintext. So any one watching this transaction now has the plaintext challenge and the encrypted response. Therefore,

3. It is acceptable to cryptographically derive two separate keys from a single master key. If done correctly, the two keys are effectively independent.

simply by XORing the two together, the enemy has a copy of the RC4 random bytes. Remember the basic equation:

```
P ⊕ R = C    (Plaintext XOR Randombytes = Ciphertext)
```

And remember that XORing twice gets you back to the original value (that's decryption):

```
If P ⊕ R = C then C ⊕ R = P
```

By the same argument, XORing the ciphertext with the plaintext gives you the random key stream:

```
If P ⊕ R = C then C ⊕ P = R
```

Whoops, the game's over! The attacker now knows the key stream corresponding to a given IV value. Now the attacker simply requests authentication, waits for the challenge text, XORs with the previously captured key stream, and returns the result with the previously captured IV.

To check the result, the access point appends the IV (chosen by the attacker) to the secret key and generates the RC4 random key stream. These will, of course, be the same bytes that the attacker worked out because the key and IV are the same as last time. Therefore when the access point decrypts the message by XORing with the RC4 key stream, surprise, surprise, it matches. The attacker is "authenticated" without ever knowing the secret key. Hopeless!

Although an attacker can get authenticated in this way, she can't then communicate because frames are encrypted with WEP. Therefore, she needs to break WEP encryption as well. However, there is even more bad news. For some of the methods of attacking encryption keys, the enemy needs a sample of matching plaintext and ciphertext. Sometimes this can be quite hard for an attacker to get, and there are various tricky methods that she might use to try to get such a sample. What a gift! The WEP authentication method provides a 128-byte sample free of charge. Worse, it is a sample of the first 128 bytes of the key stream, which is the most vulnerable to attack.[4] So not only does this approach *not* authenticate, it actually assists the enemy to attack the encryption keys. Hmmm… we better move on. Most systems today don't use the futile WEP authentication phase anyway.

4. We will look at this vulnerability in more detail in Chapter 11 on TKIP.

Access Control

Access control is, rather obviously, the process of allowing or denying a mobile device to communicate with the network. It is often confused with authentication. All that authentication does is to establish who you are; it does not follow that, because you are authenticated, you should be allowed access.

In general, access is usually controlled by having a list of allowed devices. It may also be done by allowing access to anyone who can prove he has possession of a certificate or some other electronic pass.

IEEE 802.11 does not define how access control is implemented. However, identification of devices is only done by MAC address, so there is an implication that a list of acceptable MAC addresses exists somewhere. Many systems implement a simple scheme whereby a list of allowed MAC addresses can be entered into the access point, even when you are operating without WEP. However, given the ease with which MAC addresses can be forged, this cannot be considered as a serious security mechanism.

If you can't trust the MAC address, the only thing left to WEP is the encryption key. If the mobile station doesn't know the correct WEP key, then the frames it sends will produce an ICV error when decrypted. Therefore, the frames will be discarded and, effectively, the device is denied access. This last line of defense is really all that the original IEEE 802.11 standard has to offer.

Replay Prevention

Let's suppose you are an attacker with a wireless sniffer that is able to capture all the frames sent between an access point and a mobile device. You observe that a new user has turned on her laptop and connected to the network. Maybe the first thing that happens is that the server sends her a login message and she enters her user name and password. Of course, you can't see the actual messages because they are encrypted. However, you might be able to guess what's going on, based on the length of the messages.

Later on, you notice the user has shut down and gone home. So now is your chance. Bring up your own client using her MAC address and connect to the network. As we have seen earlier, that part is easy. Now, if you are lucky, you'll receive a message to log in. Again, you won't be able to decode it, but you can guess what it is from the size. So now you send a copy of the message the legitimate user sent at that point. You are *replaying* an old message without needing to know the contents. If there were no replay protection, the access point would correctly decode the message; after all, it was originally encrypted with a valid key before you recorded it. The access point passes the message to the login server, which accepts it as a valid login. You, as an attacker, just successfully logged into the network and the server. It's not clear where you would go from there. However, from a security standpoint, this is a serious breach.

There are many other examples in which a replay attack can breach security unless the network is designed specifically to detect and reject old copies of messages. The wireless security protocol should allow only one copy of a message to be accepted. *Ever.*

By this time, it may come as no surprise to discover that WEP has no protection against replay at all. It was just not considered in the design. There is a sequence number in the MAC frame that must increase monotonically. However, it is not included in the WEP protection so it is easy to modify the sequence number to be valid without messing with the encrypted portion on the frame.

Replay protection is not broken in WEP; it simply doesn't exist.

Message Modification Detection

WEP has a mechanism that is designed to prevent message modification. Message modification can be used in subtle ways. The first thing people think about when message modification is proposed is to change the contents of the message in an obvious way. For example, changing the destination bank account number on a deposit or changing the amount transferred. However, in reality such large-scale modifications would be very hard to mount and assume that you can read the original message and effectively forge new messages.

If you are unable to decrypt the message, it is not obvious how modifying the ciphertext would be useful. However, even in this case modifications can be used to extract information. A technical paper by Borisov et al. (2001) proposed a method to exploit "bit flipping" in which a few bits of the ciphertext are changed at a time. They pointed out that the position of the IP header is usually known after encryption because WEP does not rearrange the byte positions. Because the IP header has a checksum, changing one bit in the header causes the checksum to fail. However, if you *also* change bits in the checksum, you might get a match. The researchers showed that, by flipping a few bits at a time and seeing whether the IP frame was accepted, based on whether responses came back, they could eventually decode portions of a frame.

To prevent tampering, WEP includes a checkfield called the integrity check value (ICV). We looked at this briefly in the previous section on frame formats. The idea behind the ICV is simple: compute a check value or CRC (cyclic redundancy check) over all the data to be encrypted, append the check value to the end of the data, and then encrypt the whole lot. If someone changes a bit in the ciphertext, the decrypted data will not have the same check word and the modification will be detected. The thinking is that, because the ICV is encrypted, you cannot go back and correct its value to compensate for the other changes you have made. It is only intended to provide protection to the ciphertext. If an attacker already knows the keys, he can modify the data and recompute the ICV before re-encrypting and forwarding the frame. So use of the ICV protects the ciphertext from tampering—right?

Wrong again! Borisov et al. pointed out a flaw in the logic. The CRC method used to compute the ICV is called a **linear method**. It turns out that, with a linear method, you can predict which bits in the ICV will be changed if you change a single bit in the message. The ICV is 32 bits. Let's suppose the message is 8,000 bits (1,000 bytes) and you flip bit position 5244. You can then compute which bits in the ICV will be changed as a result. It is typically not a single bit but a combination of bits that will change. Note that we used the word "change," not "set" or "clear." You don't need to know the actual value of the plaintext; you just need to know that if you flip the value of a certain bit in the data, you can keep the ICV valid by also flipping a certain combination of its bits. Unfortunately, because WEP works by XORing the data to get the ciphertext, bit flipping survives the encryption process. Flipping a bit in the plaintext always flips the same bit in the ciphertext, and vice versa.

If you've hung in through the argument in the last paragraph, you will see that, because of the fact that bit flipping survives encryption; the assumption that the ICV is protected by encryption just doesn't hold water. Its actual value is protected, but you can reliably flip its bits. And because you can work out which bits to flip corresponding to a change in the data, you can completely defeat its protection.

With a muffled thump, another of WEP's protections just hit the floor. The reality is that WEP provides no effective protection against ciphertext modification.[5]

Message Privacy

This is the big one: attacking the encryption method of WEP. We have seen that the other protections have already been stripped away; but, at the end of the day, if the encryption method holds up, then the attacker is very limited in what he can do. So far, it's just watching shadows or throwing rocks at the window; but if the encryption can be breached, the attacker is inside the house.

There are two main objectives in attacking the encryption: decode a message or get the keys. The ultimate success is to get the keys. Once an attacker has the keys, he is free to explore and look for the valuables. Possession of the keys doesn't automatically mean access to confidential information because there are other layers

5. In fact, in the general case, no integrity check word can be used successfully with RC4 unless it is created using a key generated specifically for integrity checking (as distinct from the encryption key).

of security inside, such as server passwords and operating system protections. However, the issue of network access is put aside. Furthermore, if an attacker can get the keys, he can probably go undetected, which is important to buy the time to find useful information. If an attack is detected, the WEP keys can be changed, putting the attacker back to square one.

The next best thing to getting the keys is to be able to get the plaintext. If you can get the plaintext in a reasonably fast and reliable way, you have access to a range of other types of attacks using message modification and replay. That information can also be used as a stepping-stone to getting the keys.

There are three weaknesses in the way RC4 is used in WEP and we will look at each case separately:

- IV reuse
- RC4 weak keys
- Direct key attack

IV Reuse

One of the first cryptographers to point out weaknesses in WEP was Jesse Walker of Intel. In October 2000 he wrote a submission to the IEEE 802.11 Standards Committee entitled "Unsafe at any key size: An analysis of the WEP encapsulation." This title was designed to get attention—and it did. Walker pointed out a number of potential weaknesses, but especially focused on the issue of IV reuse.

Let's quickly review how the IV is used. Instead of using a fixed secret key, the secret key is appended to a 24-bit IV value and then the combined IV/ secret is used as the encryption key. The value of the IV is sent in the frame so the receiving device can perform the decryption. One purpose of the IV is to ensure that two identical messages don't produce the same ciphertext. However, there is a second and more important purpose related to the way WEP uses XOR to create the ciphertext.

Let's suppose for a moment that there was no IV and only the secret key is used for encryption. For every frame, the RC4 algorithm is initialized with the key value prior to the start of the pseudorandom key stream generation. But if the key were to remain fixed, the RC4 algorithm would be initialized to the same state every time. Therefore the key stream produced would be the *same sequence of bytes* for every frame. This is disastrous because, if the attacker can figure out what that key stream is, he can decode every frame simply by XORing the frame with the known sequence. He doesn't need to know the key.

By adding the IV value to the key each time, RC4 is initialized to a different state for every frame and so the key stream is different for each encryption—much better. Let's review that statement because there is an implicit assumption: The IV value is *different for every frame*. If the IV is a constant value, you are no better off than in the static key case.

So we see that constant IV is useless. We can also see that using a different IV for every frame, and I mean *every* frame, is a good idea. What about the middle ground? There are a limited number of possible IVs so it is acceptable to use a different IV for most frames but eventually start reusing IVs that have been used in the past. The simple answer is that this is not acceptable—but it is precisely what WEP does.

Let's look at why IV reuse is a problem. We have said that the IV should be different for every frame. However, the original IEEE 802.11 standard did not say how it should be generated (actually it did not require that it be changed at all!). Intuitively you might think that the best approach would be to generate a random value. However, with random selection there is a good chance that you will get a repeating IV quite quickly. This is known as the **birthday paradox** (see sidebar).

In the case of IVs, it means that you are likely to get a duplicate IV sooner than you expect if you pick random values.

The best way to allocate IVs is simply to increment the value by 1 each time. This gives you the longest possible time before a repeating value. However, with a 24-bit IV, an **IV collision** (use of a previous value) is guaranteed after 2^{24} frames have been transmitted (nearly 17 million). IEEE 802.11b is capable of transmitting 500 full-length frames a second and many more shorter frames. At 500 frames a second, the IV space would be all used up in around seven hours.

In reality a collision is likely much sooner because there may be many devices transmitting, each incrementing a separate IV value and using it with the same key (assuming default keys are in use). Implementation errors can compound the problem. At least one major Wi-Fi LAN manufacturer always initializes the IV counter to 0 when the system is started up. Imagine that ten users come into work and start up their laptops. Depending on who does what, the IV counter of some will get ahead of others, but there will be a rich harvest of IV collisions to be had by an observer. IV collisions are a fact of life for WEP so let's look again at why collisions are a problem.

The Birthday Paradox

When you meet someone, there is only a 1 in 365 chance that the person has the same birthday as you. However, the chance of meeting someone with your birthday increases surprisingly fast as you meet more people. In fact, there is a 50% chance that you will find someone with a matching birthday within the first 25 people you meet. This is a surprising fact, which is probably why it is called a paradox.

If you know the key stream corresponding to a given IV value, you can immediately decode any subsequent frame that uses the same IV (and secret key). This is true regardless of whether the secret key is 40 bits, 104 bits, or 1,040 bits. To decode every message, you would have to know the key stream for every possible IV value. Because there are nearly 17 million possible IV values, that seems like a daunting task. However, it's not impossible: If you want to store a 1,500-byte key stream, for every possible IV you need a storage space of 23Gbytes—quite feasible on the hard disk of an everyday computer. With such a database, you could decode every message sent without ever knowing the secret key. However, you still have to find out all those key streams and that's not so easy.

Suppose you have captured two messages encrypted using the same IV and secret key. You know that the key stream is the same in both cases, although you don't know what it is yet. Using our simple notation:

$$C_1 = P_1 \oplus K_S \qquad \text{(Ciphertext msg1 = Plaintext msg1 XORed Keystream)}$$

and

$$C_2 = P_2 \oplus K_S \qquad \text{(KS is the same in each case)}$$

If you XOR C_1 and C_2, K_S disappears:

$$C_1 \oplus C_2 = (P_1 \oplus K_S) \oplus (P_2 \oplus K_S) = P_1 \oplus P_2 \oplus K_S \oplus K_S = P_1 \oplus P_2$$

This is true because XORing the same value twice takes you back to your original value.

So the attacker now has a message that is the XOR of two plaintexts. Is that useful? No not yet. However, some of the values of plaintext are definitely known, such as certain fields in the header. In other fields the value is not known, but the purpose is known. For example, the IP address fields have a limited set of possible values in most networks. The body portion of the text often encodes ASCII text, again giving some possible clues.

Over a period of time, if you collect enough samples of duplicated IVs, you can probably guess substantial portions of the key stream and hence decode more and more. It's like a collapsing building: Each block you knock away makes it more likely that the whole lot will fall down. It's hacker's celebrity squares in which you have some of the letters in a word and you try to guess the whole sentence. But once you succeed for a given IV, you can decode every subsequent frame using that IV and generate forged frames using the same IV. All without knowing the key.

This characteristic of WEP was worrisome and resulted in IEEE 802.11 undertaking the new security design. However, it was not considered a major threat to everyday use. After all, it would take a huge effort to decode a significant number of frames and the need for intelligence in guessing the plaintext makes it hard to create an automatic script tool. So the cryptographers cringed and the manufacturers worried, but the world went on after this attack was publicized. But there was worse to come.

RC4 Weak Keys

The fundamental part of RC4 is not encryption but pseudorandom number generation. Once we have a string of pseudorandom bytes, we can use them to encrypt the data by the XOR function. As we have seen, using this simple XOR function is a source of weakness if it is not applied correctly; but for the moment, let's concentrate on the pseudorandom sequence, or key stream.

RC4 works by setting up a 256-byte array containing all the values from 0 to 255. That is, each value from 0 to 255 appears once and only once in the array. However, the order in which they appear is "randomized." This is known as a permutation of the values. What's more, the values are reorganized continuously as each pseudorandom byte is generated so there is a different permutation of the array each time.

Each pseudorandom byte is generated by picking a single value from the permutation based on two index values, *i* and *j*, which also change each time. There are very many permutations (or arrangements) of 255 values that can be made. In fact, combined with the two indices, there are 512 * 256! (factorial) possibilities, a number too big to compute on any calculator we have, even using scientific notation.

This property of RC4 makes it very powerful despite its simple implementation. It is amazingly hard to distinguish an RC4 pseudorandom sequence from a real random sequence. RC4 has been studied by many cryptographers and yet the best known method for distinguishing an RC4 stream from true random data requires a continuous sample of 1Gbyte of the stream before it can reliably decide that the stream was generated by RC4. For WEP, of course, we already know RC4 is used, but this fact gives you some idea how effective RC4 really is once it gets going.

The phrase "once it gets going" in the last sentence is important. It signals the fact that RC4 has a potential weakness. To understand the weakness, let's quickly review how RC4 works. First it creates a table (the S-box) with all the values 0–255. Then it creates a second 256-byte table with the key, repeated over and over until the table is full. Then it rearranges the S-box based on values in the key table. This is the initialization phase. The first pseudorandom byte is generated by rearranging the S-box again and picking a byte.

The problem here is that there are not many rearrangements between the initial setup of the key table and the first pseudorandom byte. Fluhrer et al. (2001)

analyzed this fact, resulting in their now famous paper "Weaknesses in the Key Scheduling Algorithm of RC4." They showed that for certain key values, which they called **weak keys**, a disproportionate number of bits in the first few bytes of the key stream (pseudorandom bytes) were determined by a few bits in the key itself.

Let's look at this weakness in another way: Ideally if you change any one bit in the key, then the output key stream should be totally different. Each bit should have a 50% chance of being different from the previous key stream. The paper showed that this was not the case. Some bits of the key had a bigger effect than others. Some bits had no effect at all (on the first few bytes of key stream). This is bad for two reasons. First, if you reduce the number of effective bits, it is easier to attack the keys. Second, the first few bytes of plaintext are usually easier to guess. For example, in WEP it is usually the LLC header that starts with the same hexadecimal value "AA". If you know the plaintext, you can derive the key stream and start attacking the key.

There is a very simple way to avoid the weakness: Discard the first few bytes of the RC4 key stream. In other words, wait until the RC4 algorithm gets going before starting to use the output. A recommendation from RSA Labs is to discard the first 256 bytes of the key stream, but of course WEP does not do this and such a change would mean that old systems would no longer interoperate.

You might think that this is not so bad. After all, you might not be using a weak key; or if you know which keys are weak, you could avoid them, right? Think again. Remember the IV is added to the secret key. And the IV is always changing. So sooner or later, the IV guarantees that a weak key is generated. It brings tears to your eyes, doesn't it! But there's worse to come.

Direct Key Attacks

In their landmark paper, Fluhrer et al. showed that using a public IV value appended to the secret key generated a huge weakness because it allowed the attacker to wait for a potentially weak key and directly attack the key. There are two cases, one in which the IV is appended (after the secret key) and one in which the IV is prepended (before the secret key). The prepend case is the more vulnerable—and it's the relevant case for WEP, which, by now, should come as no surprise to you.

If you are interested in how the attacks work, get a college degree in mathematics and read the paper. But in overview, the idea is based on exploiting the weak key problem in the first bytes. First assume that you know the plaintext for the first few bytes, which you do for IEEE 802.11 because it is usually an IEEE 802.1LLC SNAP header. Watch transmissions looking for a weak key generated by the IV. Now you know that there is a correlation between the ciphertext, the plaintext, and the secret key bytes. There are only a limited number of possible values for the first secret key byte that could match the plaintext and ciphertext. After capturing about 60 such messages, the attacker can guess the first key byte with reasonable certainty.

The method can be tuned to attack each secret key byte in turn so eventually the entire secret key can be extracted. Note that increasing the key size from 40 bits to 104 bits only means that it takes 2.5 times longer to extract the key—in other words, the time to crack the key goes up linearly with key size rather than exponentially.

All the previous weaknesses of WEP pale into insignificance compared to this attack. Remember that extracting the keys is the ultimate goal of an attacker, and here is a method that directly extracts the keys in linear time. This attack blew apart the remnants of WEP security. Worse, because it used a fairly mechanical approach, it was feasible to create a script tool that would do the job unattended.

Within months, some "helpful" person invested their time into generating a cracker tool. Publicizing the threat was a service to everyone, but I leave it as an exercise for the readers to determine what satisfaction is obtained by the authors of tools that turn threat into reality and lay waste to millions of dollars of investment. However, the tool was published, it is available on the Internet, and attackers can use it to crack WEP systems open at will.

Summary

This chapter explains in detail how WEP works and then explains why you shouldn't use it. If you are currently using WEP, this chapter shows why you need to change. When the original IEEE 802.11 standard was published, Wired Equivalent Privacy (WEP) was included as a method to provide secure communications. However, as this chapter describes, WEP fell short of real needs in a number of areas.

The methods of key management were weak and did not scale to large networks. The key length was too small and some vendors introduced extensions to try to "improve the security." The final straw that broke the camel's back was the discovery of an attack that could successfully retrieve the secret keys by traffic monitoring.

It is said that those who don't read history are doomed to repeat it. This chapter provides the history. WEP is an interesting case study in the problems that can occur when security protocols are developed without proper review by security experts. Mostly the chapter is worth reading because it points out so many of the pitfalls that have been overcome in the new methods. Understanding WEP's failings before moving on will help you understand why the next-generation security methods are so much stronger.