



Lightweight Key Management for IEEE 802.11 Wireless LANs with Key Refresh and Host Revocation

AVISHAI WOOL*

Dept. Electrical Engineering Systems, Tel Aviv University, Ramat Aviv 69978, Israel

Abstract. The IEEE 802.11 Wireless LAN standard has been designed with very limited key management capabilities, using up to 4 static, long term, keys, shared by all the stations on the LAN. This design makes it quite difficult to fully revoke access from previously-authorized hosts. A host is *fully* revoked when it can no longer *eavesdrop* and decrypt traffic generated by other hosts on the wireless LAN.

This paper proposes WEP*, a lightweight solution to the host-revocation problem. The key management in WEP* is in the style of pay-TV systems: The Access Point periodically generates new keys, and these keys are transferred to the hosts at authentication time. The fact that the keys are only valid for one re-key period makes host revocation possible, and scalable: A revoked host will simply not receive the new keys.

Clearly, WEP* is not an ideal solution, and does not address all the security problems that IEEE 802.11 suffers from. However, what makes WEP* worthwhile is that it is 100% compatible with the *existing* standard. And, unlike other solutions, WEP* does not rely on external authentication servers. Therefore, WEP* is suitable for use even in the most basic IEEE 802.11 LAN configurations, such as those deployed in small or home offices. A WEP* prototype has been partially implemented using free, open-source tools.

Keywords: security, authentication

1. Introduction

1.1. Background

Network security is often seen as an application layer issue, to be addressed at the highest levels of the protocol stack. However, wireless LAN protocols require security components within layer 2, to protect both data confidentiality and access to the network. Specifically, hosts and access points need to be authenticated, and traffic needs to be encrypted. Unfortunately, these security requirements are not always identified early enough in the design process, leading to standards and products whose security is weaker than it should be. This is precisely the case with the IEEE 802.11 wireless LAN standard [14], which uses the WEP¹ protocol for data confidentiality.

The functionality offered by IEEE 802.11 is very attractive, letting users move about their home, office building, or their campus, while maintaining a working LAN connection. By now it is clear that IEEE 802.11 is a large scale commercial success, with thousands of installations ranging from large corporations to home users, and millions of devices sold.

Unfortunately, IEEE 802.11 has significant security problems. WEP data integrity is vulnerable to attack [5] and its authentication mechanisms may be defeated [3]. Moreover, the encryption protocol used in WEP has been severely compromised [11,31], and WEP-cracking software is widely available off the Internet (cf. [2]).

Many parties, including vendors (Lucent/Agere [24], Cisco [7], and others), and the IEEE P802.11 working group [33],

are working to rectify the known problems. Unfortunately, these efforts have to deal with IEEE 802.11's success. The huge installed-base makes it very difficult to retro-fit security fixes, because the new, security-enhanced, products, need to inter-operate with all the older devices.

1.2. Contributions

In this paper we focus on a management problem IEEE 802.11 wireless LANs suffer from. IEEE 802.11 has been designed with very limited key management capabilities, using up to four static, long term, keys, shared by all the stations on the LAN.² This design makes it quite difficult to fully revoke access from previously-authorized hosts. A host is *fully* revoked when it can no longer associate with the network access point, and more importantly, when it can no longer *eavesdrop* and decrypt traffic generated by other hosts on the wireless LAN.

The objective of this paper is to propose WEP*,³ a lightweight solution to the host-revocation problem on IEEE 802.11 LANs. The key management in WEP* is in the style of pay-TV systems, which need to add and revoke thousands of users every month: The AP⁴ periodically generates new keys, and these keys are transferred to the hosts at authentication time. The fact that the keys are only valid for one re-key period makes host revocation possible, and scalable: A revoked host will simply not receive the new keys. An advantageous side-effect of WEP* is that the impact of the [11,31] attack

² We are describing IEEE 802.11 in “infrastructure” (BSS) mode, which is its most popular mode of operation.

³ Pronounced WEP-Star. The IEEE 802.11 standard refers to hosts as “Stations”, with the acronym STA, hence “Star” for STAtion Revocation.

⁴ Access Point.

* Corresponding author. E-mail: yash@acm.org

¹ Wired Equivalent Privacy.

is reduced if the keys expire fast enough. We further propose a variant of the basic WEP*, called WEP**, which also fixes some of the authentication problems observed by [3,5].

The most natural setting for a WEP* implementation is in vendors' firmware. However, surprisingly, we demonstrate that WEP* can be implemented in software, using free, open-source, tools (cf. [9,21,32]). In this paper we describe a partial implementation on a Linux platform.

Clearly, WEP* is not an ideal solution, and does not address all the security problems that IEEE 802.11 suffers from. However, what makes WEP* worthwhile is that it is 100% compatible with the *existing* standard. It does not require new message formats, protocols, or encryption software. Unlike other solutions, WEP* does not rely on external authentication servers: The only entities involved in authentication and key management are the AP and the host. And, WEP* does not rely on public-key cryptography or infrastructure, and involves minimal changes to the wireless access point and host software.

Therefore, WEP* is suitable for use in all IEEE 802.11 LAN configurations, including the most basic devices deployed in small or home offices. Using WEP*, IEEE 802.11 systems of all sizes and complexities can substantially improve the *manageability* of their wireless LAN security. Finally, since it is fully standard-compliant, and is *not* patented, WEP* can be deployed unilaterally by any vendor that chooses to use it, and may be incorporated into open-source wireless LAN code.

1.2.1. Organization

In Section 2 we introduce the host revocation problem, survey the security mechanisms used within IEEE 802.11, known attacks against them, and proposed fixes. In Section 3 we describe WEP* in abstract terms. Section 4 explains how to implement WEP* within the confines of the IEEE 802.11 authentication protocol. Section 5 describes WEP* periodic key refresh schedule. Section 6 describes the open-source implementation. Section 7 describes the WEP** variant, and we conclude with Section 8.

2. IEEE 802.11 security problems

2.1. Host revocation

Every node on an IEEE 802.11 LAN (both APs and hosts) can be configured with up to four symmetric WEP keys, that are used for encryption and decryption of WEP-protected messages. Every node has one of its four WEP keys designated as its "default key". The default key is used to encrypt all the messages being transmitted by the node. All four keys may be used to decrypt received messages. Every WEP-encrypted message has a 2-bit header field that contains the index of the key that was used to encrypt the message.

There is no provision in the standard for distributing, refreshing, or revoking these keys. Presumably, these aspects of key management were left for the vendors to define. Unfortunately, the solutions offered by most vendors use the simplest

key management system possible: The 4 shared keys need to be manually entered into each and every device on the network.⁵

This key management approach makes it very hard to revoke a host's access. A case in point is when a user (Alice) is fired. Even assuming that her laptop and IEEE 802.11 PCMCIA card are returned to her employer, it is likely that Alice still has a copy of the company's keys. What is to prevent her from using another laptop to access the wireless LAN while sitting in the parking lot? Alice may even access the LAN from a few miles away, if she has the expertise to build a \$5-worth home-made antenna [26]. The same situation may occur when a laptop is stolen.

If the company uses MAC-based access control (see Section 2.4), the network administrator can remove Alice's MAC address from the list of allowed MACs. Unfortunately, this only ensures that she will not be able to associate with the AP, and/or will not be able to obtain an IP address on the LAN (assuming she is not sniffing MAC addresses of the air). In other words, disabling her MAC can, at best, prevent Alice from actively transmitting messages. However, since she has the WEP keys, and the same keys are used by everyone on the wireless LAN, she can *still* decrypt every message she can passively sniff.

Lacking an automated key update mechanism, the only option available to the security-conscious network administrator, who wants to fully revoke Alice, is to replace the WEP keys on each and every device on the LAN. This is a tedious, error prone, manual procedure, that is disruptive to all the legitimate users. In many cases, a typical network administrator would assume the risk and *not* change the keys, possibly for a period of weeks or months.

2.2. WEP and WEP-cracking

WEP utilizes the RC4 cipher [27] (cf. [29]), which is a stream cipher with a variable-length key. WEP uses as its RC4 key a concatenation $IV || k$, where IV is a 24-bit initialization vector, and k is one of the four shared secret keys. The IV is sent, in the clear, in the IEEE 802.11 frame. The IEEE 802.11 standard requires the IV to change between packets, but the actual IV selection method is unspecified. Basic WEP uses 40-bit keys. This rather short key length has been extended by nearly all vendors to 104-bit keys.⁶

Several weaknesses in RC4 were discovered in [11]. The most serious of these is that the first key-stream byte reveals some information about the secret key. This can be exploited by a known-plaintext attack: Given a sufficient number of WEP-encrypted packets, whose first plaintext byte is known, encrypted using the same key but with different IVs, an attacker can efficiently compute the secret key, byte-by-byte. Breaking 104-bit keys is essentially as easy as breaking 40-bit keys with this attack. This attack was successfully implemented by [31], and subsequently, WEP-cracking software became freely

⁵ Some vendors are now offering additional key management capabilities using a back-end RADIUS server via the IEEE 802.1x framework [15].

⁶ Misleadingly advertised as "128-bit encryption": The 24 IV bits are not secret.

Seq.	Direction	Payload	
#1:	host → AP	: -	// authentication request implied
#2:	AP → host	: N	// "challenge text"
#3:	host → AP	: WEP[N]	// Encrypted challenge
#4:	AP → host	: "successful"/"unsuccessful"	

Figure 1. The IEEE 802.11 authentication protocol.

available off the Internet (cf. [2]). The attack needs between 1,000,000–5,000,000 packets to succeed.

2.3. Authentication: Identity faking and key-stream exposure

IEEE 802.11 uses a 4-message authentication protocol. Messages are identified by their *sequence number*, which is a number between 1–4 (see figure 1). The communicating parties are the AP, and the host that is attempting to associate with the AP.

Notice that message #3 is the WEP-encryption of the challenge text N that was transmitted in message #2. In other words, N is encrypted using RC4, using one of the four keys in the WEP key-set, combined with an IV. The IV is transmitted, in the clear, as part of message #3. Note also that messages #2 and #3 do not have the communicating parties' identities inside their payloads, and do not have any data integrity protection beyond the general CRC-32. Message #2 does not have any freshness field associated with the challenge N .

The weaknesses in this protocol were highlighted in [3]. The most serious is that since RC4 is a stream-cipher, an eavesdropper, Eve, can sniff both message #2 and message #3, and XOR them together. This produces 128 bytes of RC4 PRNG key-stream, that are the output of RC4 used with the (unknown) WEP key k combined with the (known) IV. Having this key-stream and IV, Eve can forge messages to the LAN. If there is no MAC-based access control (Section 2.4), Eve can also authenticate herself to the AP by responding correctly to the AP's challenge message #2: She can fix the IV to the IV value she knows, and use the exposed key-stream to encrypt N .

2.4. MAC-based access control

Although not part of the IEEE 802.11 standard, many vendors (e.g. Agere [24], Cisco [8], Linksys [18]) have added MAC-based access control to their solutions. This access control usually takes the form of a list of MAC addresses that are allowed to associate with an AP. When a host sends an authentication-request message (message #1) to the AP, the AP looks up the host's MAC address in the list, and proceeds with the authentication protocol only if the MAC is allowed.

While MAC-based access control is useful, it cannot repel a determined attacker, for two reasons. First, MAC addresses may be modified fairly easily by users. Second, the MAC addresses are transmitted in the clear in every IEEE 802.11 frame, so they can be sniffed. Thus, MAC access control does not prevent the identity-faking attack of [3] (Section 2.3): An attacker can sniff an allowed MAC address off the air, spoof her MAC address to the allowed address, and associate with the AP.

2.5. Compromised data integrity

The IEEE 802.11 standard does not have a cryptographic mechanism to defend its data's integrity. Data integrity is only protected by CRC-32. Unfortunately, as noted by [5], CRC-32 provides absolutely no data integrity against a malicious attacker, even when both the data and CRC are covered by WEP encryption. CRC-32 is an unkeyed linear function of the data. WEP encrypts by XORing the RC4 key-stream with the data, and XOR is a linear operator as well. Thus, an attacker that sniffs a WEP-encrypted message M can easily flip any bit positions of her choice in M , and she can adjust the message's CRC-32 code to match the modified message. This can be done, through the RC4 encryption, by XORing easily computed bit strings with M .

2.6. Proposed fixes

In light of the [11,31] WEP-cracking attacks, the IEEE 802.11 TGi working group is considering using the AES cipher in CBC mode [1], instead of RC4, in a future revision to the standard. The working group is also considering the use of a *keyed* hash function (a MIC⁷) [10] instead of CRC-32. These changes will, if adopted, defend against the [11,31] attacks, the [5] attacks, and some of the attacks of [3].

In the interim, and to support exiting equipment, several other suggestions have emerged, that attempt to address the problems without decommissioning RC4. Most rely on the fact that the [11,31] attacks require the collection of several million packets that are encrypted using the same WEP key with different IV values. One way to defend against the attack is to change the WEP key frequently, thus denying the attacker from collecting enough encrypted data. Both [36] and [6] suggest that the AP and host share a long term master secret, from which they derive the WEP keys periodically. The IEEE 802.11 TGi working group is considering this approach for use by legacy equipment, as part of the proposed TKIP⁸ mechanism. Note that this type of re-keying defends against WEP-cracking (if the keys are changed fast enough), but does nothing to support host revocation: A revoked host that holds the master secret can continue to derive the WEP keys indefinitely.

Several systems address the authentication problems of [3]. Major vendors [7,23] are deploying systems that use an external authentication server, instead of the AP, to authenticate hosts. The authentication server is usually a RADIUS server, typically running on a Microsoft Windows platform. The authentication protocol uses the IEEE 802.1x authentication framework [15], which has some security problems of its own [19]. RADIUS-based authentication via 802.1x is also under consideration by the IEEE 802.11 TGi working group for possible inclusion in the next revision of the IEEE 802.11 standard. Some vendors' systems also include a key transport mechanism, where the RADIUS server provides the

⁷ Message Integrity Code.

⁸ Temporal Key Integrity Protocol.

WEP keys to the host after the host is authenticated. A related approach [28] integrates the authentication into a modified DHCP server. Clearly, these solutions only apply to organizations that are willing to install and manage an additional authentication server. As such, these solutions do not help the vast numbers of home and small-office installations. Furthermore, solutions that do not include a key transport mechanism do not support full host revocation: A revoked host may not be able to authenticate itself, but it will still be able to decrypt traffic.

3. WEP*: Key management with host revocation

3.1. WEP* design rationale

To use WEP encryption, the AP and hosts on the LAN need to maintain a shared set of WEP keys, which we denote by K_{curr} . To allow scalable host revocation, though, K_{curr} clearly cannot be static. In WEP*, it is the AP's responsibility to periodically refresh K_{curr} . If the key-set changes over time, host-revocation becomes feasible: Revoking the credentials of Alice's laptop from the AP ensures that she will not get any future key updates. After the last of Alice's WEP keys expires, her access to the LAN will be fully revoked.

WEP* uses the IEEE 802.11 authentication protocol to securely *transport* the K_{curr} key-set to the hosts. Note that the IEEE 802.11 authentication protocol runs whenever a host joins the LAN, so every new host will get the latest K_{curr} . However, periodically we also need to get hosts that are already associated with the AP to refresh their key-set. We shall deal with this issue in Section 5.

To secure the key transport protocol, we require every host on the LAN to share a long-term secret with the AP, which is made of two keys: an encryption key which we denote by k_{host} , and a message integrity code (MIC) key which we denote by k_{mic} . Each host has its own unique pair of keys. The host key k_{host} is not used to encrypt data messages, it is only used within the authentication protocol to prove the identity of the host and to encrypt the current WEP key-set K_{curr} . Therefore, k_{host} may be viewed as a "key-encryption-key". Likewise, the MIC key k_{mic} is only used to ensure the integrity of the authentication messages.

The AP needs to maintain a mapping $\text{HostID} \mapsto \{k_{\text{host}}, k_{\text{mic}}\}$, mapping a host identifier to the key pair shared with that host. WEP* uses the MAC address of a host as its HostID. Recall that almost all vendors' APs are capable of maintaining a list of allowed MAC addresses: So all that is needed to maintain the host-to-key mapping are two additional key fields in the allowed-MAC-address table.

As we mentioned in Section 2.4, MAC addresses are unreliable HostIDs. However, MAC addresses are not viewed as authoritative identifiers in WEP*. Rather, they have the same semantics as a "username" in a login procedure. Namely, the MAC indicates who the host *claims* to be, and this identity is proved by the host's possession of the shared secret key k_{host} . Thus, the security of WEP* does not rely on MAC addresses being immutable or secret.

There are many ways to implement the type of key management we just described. However, the foremost design goal of WEP* is to comply with the IEEE 802.11 standard, and to be implementable on existing systems. This goal severely constrains the cryptographic primitives and protocols we can use. For instance, public-key cryptography is immediately ruled out: Public-key computations are prohibitively expensive for the weak processors in IEEE 802.11 network interface cards.

3.2. Point-to-point key update

Assume that the host and the AP share the long-term secret keys k_{host} and k_{mic} , and that the AP has the current WEP key-set K_{curr} . The host already contacted the AP to start the IEEE 802.11 authentication protocol. What we need now is a secure mechanism to transport K_{curr} from the AP to the host.

This is a well studied cryptographic protocol requirement, and several solutions are described in [22, ch. 12.3.1]. Among them, the solution that is best suited for use in WEP* is the "point-to-point key update protocol" [17]. Other possible solutions involve either a key distribution center (KDC) or public-key cryptography, neither of which seems appropriate for WEP*. Specifically, we use a one-pass protocol, from the AP to the host. We use the following notation:

- $K_{\text{curr}}.k[0], \dots, K_{\text{curr}}.k[3]$ are the four WEP keys in the current key-set. $K_{\text{curr}}.defkey$ is the index selecting the default (transmission) key, $0 \leq K_{\text{curr}}.defkey \leq 3$.
- Δ is an upper bound on the maximum time difference between the clocks of the host and the AP.
- $E_k[M]$ is the RC4 encryption of message M using key k . There is no IV, and k forms the entire RC4 key. We assume that k is at least 64-bit long.
- t_{AP} and t_{host} are timestamps generated by the AP and host respectively. MAC_{AP} and MAC_{host} are the MAC addresses of the AP and host respectively. $H(k_{\text{mic}}, M)$ is a MIC of a message M using the MIC key k_{mic} .

The point-to-point key update protocol involves a single message:

$$M \stackrel{\text{def}}{=} (K_{\text{curr}}, t_{\text{AP}}, \text{MAC}_{\text{AP}}, \text{MAC}_{\text{host}})$$

$$\text{AP} \rightarrow \text{host} : E_{k_{\text{host}}}[M, H(k_{\text{mic}}, M)] \quad (1)$$

Upon receipt of message (1), the host decrypts the message using k_{host} . Denote the received fields by $(K_{\text{curr}}, t_{\text{AP}}, \text{MAC}_{\text{from}}, \text{MAC}_{\text{to}}, h)$. The host checks the message integrity by verifying that $H(k_{\text{mic}}, \{K_{\text{curr}}, t_{\text{AP}}, \text{MAC}_{\text{from}}, \text{MAC}_{\text{to}}\}) = h$, verifies that $\text{MAC}_{\text{to}} = \text{MAC}_{\text{host}}$, $\text{MAC}_{\text{from}} = \text{MAC}_{\text{AP}}$ (where MAC_{AP} is the MAC of the AP the host wanted to associate with) and that $|t_{\text{host}} - t_{\text{AP}}| \leq \Delta$. If all the checks were successful, the host accepts K_{curr} as the current WEP key-set.

Remarks.

- The MIC $H()$ protects the integrity of the message from modification "through" the RC4 stream-cipher encryption

[5] (recall Section 2.5). Unlike CRC-32, $H()$ is not a linear function. The IEEE 802.11 TGj working group is considering using MICs in the next version of the standard (cf. [10]).

- In addition to securely transferring K_{curr} to the host, the one-pass protocol authenticates the AP to the host, and prevents replay attacks against the host. However, it does not authenticate the host to the AP, nor does it protect the AP from replay attacks or from the forgery and impersonation attacks of [3]. We can defend against the attacks of [3] by using a two-pass key transport protocol—however, we have not found a 100% IEEE 802.11-compliant way to do so. Our best proposal for a two-pass protocol, called WEP*(Section 7), is almost, but not quite, IEEE 802.11-compliant.

4. WEP*: Embedding key management within IEEE 802.11

4.1. What can we use

Our goal is to allow host-revocation and key refresh, and to do so strictly within the framework of the IEEE 802.11 standard. We identify two components of the standard which offer some flexibility that we can utilize for our purposes.

4.1.1. The “challenge text” message

The IEEE 802.11 authentication message #2, sent from the AP to the host, contains a “challenge text”, which is surprisingly long: 128 bytes. This challenge text is really just a nonce. The purpose of a nonce (and a response containing its encryption) is to prove to the sending party that the response is fresh and not a replay. To prove freshness, though, a 64 bit nonce is usually quite adequate. It is unclear why the standard requires 128 bytes (= 1024 bits) of nonce.

The IEEE 802.11 standard recommends that the challenge text should be pseudo-random. However, the only formal requirement from it [14, sec. 8.1.2.2] is that it must not be comprised of a constant byte repeated 128 times. Other than this requirement, the challenge text may essentially be arbitrary. Therefore, we can add more semantics to the “challenge text” message. In particular, WEP* uses the “challenge text” field to transport the current key-set K_{curr} from the AP to the host, encrypted by the host key k_{host} .

4.1.2. Accurate synchronized clocks

Point-to-point key transport protocols (Section 3.2) normally require a freshness field in the messages, to prevent replay attacks. The freshness field could be a random nonce, a sequence counter, or a timestamp. Sequence numbers require the com-

municating parties to keep state (for the last sequence number). Timestamps require tightly synchronized clocks, which are usually hard to guarantee in high levels of the protocol stack. Therefore, the most frequently used freshness field is a random nonce.

However, since IEEE 802.11 is a layer 1 and 2 (PHY+MAC) standard, it requires very accurate and *synchronized* clocks for hardware framing and slotting. Both the AP and host are required to maintain 64-bit, microsecond-precision, clocks. These clocks are guaranteed to be synchronized “to within 4 microseconds plus the maximum propagation delay of the PHY for PHYs of 1 Mbit/s, or greater” [14, Sec. 11.1.2]. The host synchronizes its clock to the AP’s clock before any authentication takes place, either by listening to Beacon messages sent by the AP, or by sending a Probe message and listening for the Probe-Response message. Therefore, it is both straightforward and reasonable to use a timestamp as a freshness field in IEEE 802.11 authentication.

We assume that the attacker does not have the ability to modify the AP’s clock. Having such access would imply that the attacker has physical access to the AP and is able to tamper with it—in which case one can assume the attacker can also read the secret keys.

4.2. WEP* details

As we mentioned in Section 3.1, we assume that the host and AP share a long term secret key pair $\{k_{host}, k_{mic}\}$, and that the AP has a host-to-key lookup table. We also assume that the AP already has a WEP key-set K_{curr} .

WEP* authentication has a message flow which is identical to normal IEEE 802.11 authentication (compare with Section 2.3). The main protocol change we make is that we modify the content of the “challenge text” field in message #2 in the authentication sequence. Instead of a random nonce, this field will now contain an instance of the one-pass point-to-point key update message (1) (Section 3.2).

The authentication protocol begins by the host sending message #1 to the AP, requesting authentication.

4.2.1. Operations at a WEP* AP

Upon receipt of message #1, the AP looks up MAC_{host} . If the host is allowed to receive service, the AP obtains k_{host} from the lookup table. The AP then prepares the message body M , specified in figure 2. The t_{AP} , MAC_{AP} , and MAC_{host} fields have the meanings we introduced in Section 3.2. The 1-byte “keylen” field contains the length of the WEP keys (possible values: 5 or 13). The 1-byte “defkey” field contains $K_{curr}.defkey$, and $k[i]$ contains $K_{curr}.k[i]$ for $i = 0, \dots, 3$. If $keylen = 5$ then the rightmost 8 bytes in every 13-byte $k[i]$ field are ignored, and their content may be arbitrary. The 8-byte

Field:	H	t_{AP}	MAC_{AP}	MAC_{host}	T_{rekey}	keylen	defkey	$k[0]$	$k[1]$	$k[2]$	$k[3]$
Bytes:	20	8	6	6	8	1	1	13	13	13	13

Figure 2. The format of the message M , which is encrypted and transmitted as the “challenge text” in IEEE 802.11 authentication message #2.

“ T_{rekey} ” field contains the re-key period, which is discussed in Section 5.

The AP first fills all the fields other than H in M (see figure 2). The AP then computes the MIC $h = H(k_{\text{mic}}, M)$ of those fields. For concreteness, we use HMAC-SHA1 [4,30] as the MIC (20 output bytes). The AP then places h in the leading H field in M , and encrypts the result using k_{host} as the key, to produce the encrypted message $C = E_{k_{\text{host}}}[M]$. There is no IV in this encryption operation.

The encrypted C is 102 bytes long. To produce a 128-byte challenge text, the AP adds 26 padding bytes after C . The padding bytes are not encrypted under the k_{host} key, and should create a fixed pattern. For concreteness, we suggest that all the padding bytes be “stars” (“*”, hexadecimal value of 0x2a). The final 128-byte challenge text N is:

$$N = E_{k_{\text{host}}}[M], *, *, \dots, *. \quad (2)$$

The AP sends the IEEE 802.11 authentication message #2 to the host, with the challenge text N as its payload. This message is not marked as a WEP message.

Remarks.

- Note that we placed the hash value h as the first field in M , inside the encrypted C . This is because the RC4 weakness of [11] relies on having known plaintext in the first byte positions. Placing the pseudo-random message hash in the first byte positions makes the [11] weakness harder to exploit.
- The T_{rekey} field does not have to be part of the secure key transport protocol. We include it in the message format for convenience. It could also be transported from the AP to the host using insecure methods, such as in a modified Beacon message (cf. [6]).

4.2.2. Operations at a WEP* host

Upon receipt of message #2, the host checks whether the last 8 bytes of the challenge text N all have the value “*”. If not, the host concludes that the AP is not using WEP*. In such a case the host continues to use the WEP key-set it already has installed (presumably entered manually). If the last 8 bytes of N all have the value “*”, the host takes the actions outlined in Section 3.2. Namely:

1. Decrypt N (ignoring the padding) using the key k_{host} . Denote the value of a field F within the message M by F^M .
2. Check the message integrity by verifying that $H(k_{\text{mic}}, M) = h^M$.
3. Defend against replays by verifying that $\text{MAC}_{\text{host}}^M = \text{MAC}_{\text{host}}, \text{MAC}_{\text{AP}}^M = \text{MAC}_{\text{AP}}$ (where MAC_{AP} is the MAC of the AP the host wanted to associate with) and that $|t_{\text{host}} - t_{\text{AP}}^M| \leq \Delta$ (recall that Δ is an upper bound on the maximum absolute time difference between the clocks of the host and the AP, and t_{host} is the host’s current clock value).

4. If one or more of the tests failed, the host aborts the authentication protocol with a failure.
5. If all the tests are successful, the host installs the values in the $k[0]^M, \dots, k[3]^M$ fields into its WEP key-set K_{curr} .
6. Set $K_{\text{curr}}.\text{defkey} \leftarrow \text{defkey}^M + 1 \pmod{4}$. See Section 5 for an explanation why incrementing defkey is necessary.
7. Install the value of the T_{rekey}^M field into a re-key-period state variable, and copy the current value of the t_{host} clock into a key-install-time state variable.

At this point, the host’s K_{curr} contains a fresh copy of the AP’s WEP key-set, so the host can continue the IEEE 802.11 authentication protocol as usual. Specifically, the host sends authentication message #3 back to the AP. This is a WEP-encrypted message whose payload is the raw challenge text N . The AP should then respond with authentication message #4, thereby completing the IEEE 802.11 authentication.

Remarks.

- Since we have not made any change to authentication message #3, it is still susceptible to the attacks of [3] (recall Section 2.3). However, since the K_{curr} changes over time, the attacker, Eve, will need to repeat her attack periodically. Furthermore, Eve cannot decrypt any LAN traffic since she does not know K_{curr} . So, in essence, Eve can mount a denial-of-service attack on the wireless LAN, or she can mount some of the more sophisticated attacks suggested in [5].
- Authentication message #3 is encrypted using the WEP key $k' = K_{\text{curr}}.k[K_{\text{curr}}.\text{defkey}]$, combined with a host-chosen IV. The probability that $\text{IV} \parallel k' = k_{\text{host}}$ (which would cause message #3 to be plaintext) is a negligible 2^{-64} . However, this rare condition can be tested, and if it occurs the host can pick a different IV.
- The probability that a non-WEP* AP will send a challenge text N , whose final 8 bytes are “*”, is a negligible 2^{-64} , since the IEEE 802.11 standard recommends using RC4 PRNG output for the challenge. If this event does happen, the host will attempt to decode the challenge text, will fail the hash verification, and abort.
- We assume that the host knows the maximum clock difference Δ . Δ should be based on a hard-coded value of 4 microseconds, plus a term that depends on the PHY speed (per [14, sec. 11.1.2]), plus an upper bound on the MAC layer processing time.

5. WEP* periodic key refresh

5.1. Operations at the AP

In WEP* it is the AP’s responsibility to generate and refresh the current WEP key-set K_{curr} . The AP achieves this by incrementally generating an unpredictable sequence of WEP keys w_0, w_1, \dots, w_j , etc. At any given time, the K_{curr} key-set

contains a window of 4 keys, w_j, \dots, w_{j+3} , for some value of $j \geq 0$.

The keys in the sequence $\{w_j\}$ should form a pseudo-random sequence. A possible generation method can use a keyed hash function $F_z()$ with a secret master-key z (that only the AP knows). When it needs a new key w_j , the AP evaluates F_z on the concatenation of the previous keys and several time-varying parameters it has access to (e.g., number of packets sent/received, number of beacons transmitted, number of probes received, signal/noise ratios, current clock time, etc.).

At power-up time, the AP generates the first four keys w_0, \dots, w_3 , and sets $K_{curr}.k[i] \leftarrow w_i$ for $i = 0, \dots, 3$. The initial value for the default key is $K_{curr}.defkey \leftarrow 0$. Subsequently, at the beginning of each re-key period, the AP performs the following actions:

1. Generates a new key w_{j+4} .
2. $K_{curr}.k[K_{curr}.defkey] \leftarrow w_{j+4}$.
3. $K_{curr}.defkey \leftarrow K_{curr}.defkey + 1 \pmod{4}$.

This guarantees that at any point in time, K_{curr} contains the key w_j which the AP is using for transmission (the AP's default key), together with the keys w_{j+1}, \dots, w_{j+3} that the AP will use for transmission during the next 3 re-key periods. It is easy to see that the AP's default key is always the "oldest" key in the K_{curr} key-set. Note that the order of operations 2 and 3 is important.

5.2. Operations at the host

When a host authenticates itself to the AP, as described in Section 4.2.2, it receives a copy of the K_{curr} key-set held by the AP, and the duration of the re-key period T_{rekey} , which is kept in the re-key-period state variable. The host also records the value of its clock at the time it received the key-set from the AP (denoted by $t_{install}$) in its key-install-time state variable. At times $t_{install} + T_{rekey}$ and $t_{install} + 2T_{rekey}$ the host performs a local re-key operation by setting

$$K_{curr}.defkey \leftarrow K_{curr}.defkey + 1 \pmod{4}. \quad (3)$$

Note that $t_{install}$, the time at which the host received the key-set from the AP, is typically not the beginning of the AP's re-key period. Suppose the 4 keys the host received at time $t_{install}$ are the keys w_j, \dots, w_{j+3} . Then at some time t , $t_{install} < t < t_{install} + T_{rekey}$, the AP will refresh its keys to w_{j+1}, \dots, w_{j+4} , evict w_j from its key-set, and start using w_{j+1} as its transmission key. After this occurs, the host will still be able to decrypt the AP's messages, because the host has w_{j+1} . However, the AP will not be able to decrypt any message that the host encrypts using w_j , because this key has just been evicted by the AP. For this reason, the host needs to increment $K_{curr}.defkey$ when it receives a key-set from the AP (see Section 4.2.2, step 6). Doing so ensures that the host will only use the three keys w_{j+1}, \dots, w_{j+3} as default (transmission) keys, so the AP will be able to decrypt all the host's message.

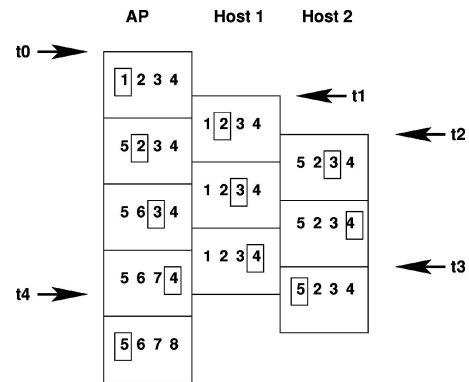


Figure 3. A sketch of WEP* key refresh. Each column represents one node, with time progressing downward, and each box corresponding to a re-key period. The numbers represent the indices of the keys in the sequence $\{w_j\}$. For each node and each re-key period, the index of the default key is shown in a rectangle. The AP re-keys by generating a new key and incrementing its default key. The hosts only increment their default key values. Host 1 authenticates itself at time t_1 and Host 2 authenticates itself at time t_2 . It is easy to verify that Host 1 and the AP can communicate for 3 full re-key periods, between times t_1 and t_4 . However, Host 1 can communicate with Host 2 only during the first two of Host 1's re-key periods: At time t_3 Host 2 starts encrypting with key 5, which Host 1 does not have.

Remarks.

- Even though the host does not use the oldest key w_j for transmission, it still needs to have a copy of w_j so it can decrypt the AP's messages prior to the AP's re-key.
- At time $t_{install} + 3T_{rekey}$, the host's key set will expire. At that time the host would complete using w_{j+3} as its transmission key, but it would not have w_{j+4} . Therefore, to ensure continued connectivity with the AP, the host would need to re-authenticate itself to the AP.
- A host is guaranteed to be able to communicate directly with other hosts on the LAN only during the first 2 re-key periods starting at time $t_{install}$ (see figure 3). Such direct host-to-host communication can occur in "Ad Hoc" (IBSS) mode, which is sometimes employed even when an AP exists. If this mode of communication is important, then the host needs to re-authenticate itself to the host before time $t_{install} + 2T_{rekey}$.

5.3. Re-authentication

When the host's key-set expires, it needs to re-authenticate itself to obtain a fresh key-set. There are several possible mechanisms to trigger a host's re-authentication event:

- When a host's keys are about to expire, the AP can send the host a Deauthentication message. However, this would require the AP to keep state for each authenticated host. Furthermore, this approach could lead to a "re-authentication storm" at the end of every re-key period, unless care is taken to spread the hosts' re-authentication over the whole re-key period.

- The host can simply use the key-set without any re-authentication. At some point the key-set will expire. This will drop the user's LAN connection, causing the user to manually reset the connection and re-authenticate. This disruptive "mechanism" is the one which non-WEP* hosts will experience.
- The preferred re-authentication mechanism is for the host itself to detect when its keys are about to expire (between times $t_{\text{install}} + 2T_{\text{rekey}}$ and $t_{\text{install}} + 3T_{\text{rekey}}$). The host will then send a Deauthentication message to the AP, and proceed to re-authenticate itself.

5.4. Compatibility with non-WEP* equipment

As described in Section 4.2.2, if the AP is not using WEP* but the host is, the host will revert to pure IEEE 802.11 authentication, so the compatibility would be perfect, assuming that the host's WEP key-set is entered manually.

The reverse situation, of a WEP*-AP working with a non-WEP* host, is more interesting. The authentication message flow between the host and AP will be the usual IEEE 802.11 authentication. Obviously, the host's WEP key-set has to be manually entered. However, at some point the key-set will expire, since the AP will have refreshed its keys, so the user will need to obtain a new key-set and re-enter it manually.

Therefore, in a heterogeneous environment, the re-key period will have to be chosen to balance user convenience against the system's security. For instance, picking a re-key period of 3 hours would give non-WEP* users roughly 9 hours of uninterrupted work. The risk with such a long period is that if a host key is revoked, the host can still use its current WEP key-set for a relatively long period of time to use the key. Therefore, the length of the re-key period is a trade-off between the system's speed of revocation and the inconvenience to non-WEP* users.

Note that the re-key period recommended in [7] is 10–20 minutes. This short period is supposed to aid in combating the [31] attack. However, the purpose of WEP* was to allow host revocation, and improving the resistance to WEP-cracking is only a fortuitous side-effect of the system rather than its main goal.

6. Open-source implementation

WEP* requires changes in the way the AP and host handle IEEE 802.11 authentication messages. These messages are MAC-layer management frames, and are typically handled by the vendors' firmware. Therefore, it is somewhat surprising that a software-only open-source implementation of WEP* is at all possible.

Luckily, IEEE 802.11 cards based on the Intersil Prism 2/2.5/3 chip set [25] have a "HostAP" mode. In this mode, the card's firmware hands off all but the most time-sensitive layer-1 operations to the driver software. An open-source driver, also called HostAP [21], uses this capability of the Prism cards. With the HostAP driver, one may create an

IEEE 802.11 AP out of a PC running Linux. This environment allows us to experiment with software modifications to the layer-2 protocol without vendor support.

6.1. Experimental hardware and software

For the AP we used a Compaq Proliant 400 PC, running RedHat Linux 7.3 with kernel version 2.4.18-3 (this is the default kernel for RedHat 7.3, no recompilation was necessary). The wireless card was a Linksys WMP11 Wireless PCI Card (which is based on the Prism 2.5). To support the HostAP mode correctly, the card's firmware needs to be upgraded to version 1.4.9. We used the latest development version of the HostAP driver (as of July 24, 2002), from the top of the developer's CVS tree [21]. We also used Wireless Tools v.24 [32].

6.2. Key refresh

The WEP* AP key refresh mechanism can actually be implemented outside the HostAP driver code. The driver supports `ioctl` calls to set the WEP keys and the default key. These `ioctl` calls can be activated from user-mode via the Wireless Tools `iwpriv` utility, or the HostAP `hostap_crypt_conf` utility.

In our implementation, the current WEP key-set is maintained in a file. The algorithm described in Section 5.1 is implemented as a program (`WEPS_genkey`) that reads the key file, generates one new WEP key, computes the new index of the default key, and rewrites the key file. The new key is generated pseudo-randomly using the Cryptlib library's key generation functions [13]. `WEPS_genkey` is executed at intervals of T_{rekey} via a cron job, which also uses the `hostap_crypt_conf` utility to install the new key-set in the driver. We currently use a re-key period of 3 hour, giving a window of 9 hours of uninterrupted connectivity to legacy hosts (recall Section 5.4).

6.3. Layer-2 modifications

The HostAP driver includes a function (`ap_auth_make_challenge`) that produces an IEEE 802.11 challenge using the RC4 cipher. It is straightforward to replace the challenge creation algorithm with the method described in Section 4.2.1, with one caveat. The synchronized layer-1 clock is maintained in the Prism firmware, and it is unclear whether it may be read by the driver [20]. A crude approximation may be obtained by using the un-synchronized clock on the AP. The HostAP driver does support MAC-based access control, which is controlled via `ioctl`. Thus it is not hard to extend the `ioctl` interface to support an additional key per MAC address.

The situation on the host side is more complex. As of this writing, the HostAP driver does not support host functionality (i.e., it does not generate authentication messages #1 and #3, and more importantly, it does not respond to authentication message #2). However, this functionality is on the HostAP developer's to-do list [20], so it is safe to assume that it can be built in software. Although several other open-source host-side IEEE 802.11 drivers are available for Linux, we are not

currently aware of one that performs WEP authentication outside the firmware. This is the subject of ongoing research.

Without the ability to modify the host-side authentication operations, we were not able to completely test WEP* beyond its key-refresh operations. Therefore, the AP currently in operation effectively treats all hosts as non-WEP* hosts. Nevertheless, a clear implementation path does exist for open-source software-only host-side WEP* based on Prism cards.

7. Preventing replay and forgery attacks using WEP**

Recall that the [3] attack exploits the fact that payload of authentication message #3 is the encryption of the nonce *N* under the current WEP key. The nonce *N* was sent in the clear in authentication message #2. The combination of the two messages leads to key material exposure. Unfortunately, this message structure is required by the IEEE 802.11 standard. Therefore, the WEP* protocol we have seen is still vulnerable to the [3] attack, since, by design, it is 100% compliant with the standard.

The goal of the WEP** variant is to defend against the [3] attack, but to remain very close to the standard. In particular, a design goal is not to change the IEEE 802.11 authentication message flow. Therefore, the only change we propose in WEP** (beyond WEP*) is to modify the payload of authentication message #3.

In WEP**, message #3 carries a WEP encryption of something which is *not* the raw nonce *N*. Instead, the host assembles a message *D* (see figure 4), WEP-encrypts it using the current key-set (which has been installed after receiving authentication message #2), and sends the result to the AP. Message *D* contains the AP’s timestamp *t*_{AP} (which the host copies from the decrypted message #2), the MAC addresses of the host and AP, and the host’s own current timestamp. Message *D* is also prepended by a MIC of its body, for the same reasons discussed in Section 4.2.1. Message *D* is 48 bytes long.

When the AP receives message #3 from the host, it can detect that this is a WEP** message by its length: A regular IEEE 802.11 authentication message #3 has a length of 128 bytes. The AP then decrypts the message, and verifies the following conditions: (1) the MIC *H* matches the contents of *D*; (2) the received *t*_{AP} field matches the timestamp which the AP placed in message #2; (3) the received MAC addresses match the AP’s own MAC and the authenticating host’s MAC; (4) the timestamp *t*_{host} is recent, $|t_{\text{host}} - t_{\text{current}}| \leq \Delta$. If all the conditions hold, the AP completes the authentication successfully and sends authentication message #4 to the host.

Field:	<i>H</i>	<i>t</i> _{AP}	<i>t</i> _{host}	MAC _{AP}	MAC _{host}
Bytes:	20	8	8	6	6

Figure 4. The format of the message *D*, which is encrypted and transmitted in WEP** as authentication message #3.

Remarks.

- A better solution would have performed the validity tests when the AP receives authentication message #1 from the host, rather than after message #3: By the time message #3 is received, the host already learned the WEP keys. Unfortunately message #1 has no payload, and we constrained ourselves to not modify the message formats in the design goals of WEP**.
- A passive attacker can guess a significant portion of the payload of *D*: She knows the MAC addresses of the parties and can guess most of the timestamp bits based on the contents of the Beacon messages. However the random padding ensures that the attacker will have a very low probability of guessing the hash *H*. Thus the attacker would have a “crib” of about 28 bytes at the end of the message, which is not hard to obtain from many other encrypted messages. Since the exposed key material is rather short, and does not begin at the first byte position, the attacker will not be able to send messages to the LAN (i.e., the attack of [3] will not work). Furthermore, the attacks of [11] do not apply when the known plaintext is not in the first byte positions.

8. Conclusions

We have presented WEP*, which provides key management with host revocation to existing IEEE 802.11 wireless LAN networks. WEP* is quite simple, very efficient, uses well understood key transport protocols and cryptographic primitives, requires no additional equipment beyond the AP and hosts, and is not patented (nor will it be). Most importantly, though, we have shown how to incorporate WEP* into the existing IEEE 802.11 standard’s authentication messages in a way that is 100% standard-compliant. We have demonstrated a partial implementation of WEP* using open-source tools, and identified a development path for a full implementation.

Notes added in press: By the time this article went to press, the IEEE 802.11i draft [16] was ratified by the TGn task group. It includes three new modes of operation: two based on the AES cipher, and one (TKIP) still based on RC4. The TKIP mode, which is intended for use on weak legacy hardware, uses a new keyed MIC called “Michael”. Recently this MIC was shown to be fragile, to the point of breaking down completely if two messages are ever encrypted using the same IV [35]. However, TKIP uses a 48-bit IV field, and a new method of setting up the RC4 key, that is designed to eliminate the possibility of IV reuse.

Authentication is to be performed through a RADIUS server via the IEEE 802.1x authentication framework [15]. This allows dynamic WEP key generation, and thus, allows for host revocation. The path taken by the HostAP open-source project in response to this development is to interface with an open-source RADIUS server (freeradius, [12]),

that is co-located on the same Linux machine running the HostAP code. We believe that the WEP* solution is much more lightweight, and would have been easier to configure on a small installation.

Looking back at the development of IEEE 802.11, one may wonder why the security aspects of standard had so many problems. One possible, cultural, explanation is provided in [34]: security analysis, and cryptanalysis, are fundamentally different from other engineering tasks, and the normal standardization process may be at odds with the human goals of the cryptanalysts.

References

- [1] Advanced encryption standard, National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001).
- [2] The AirSnort homepage (2002). <http://airsnort.sourceforge.net>.
- [3] W.A. Arbaugh, N. Shankar, Y.C.J. Wan and K. Zhang, Your 802.11 wireless network has no clothes, *IEEE Wireless Communications* 9(6) (2002) 44–51.
- [4] M. Bellare, R. Canetti and H. Krawczyk, HMAC: Keyed hashing for message authentication, Internet Engineering Task Force RFC 2104 (Feb 1997). <http://www.ietf.org/rfc/rfc2104.txt>.
- [5] N. Borisov, I. Goldberg and D. Wagner, Intercepting mobile communications: The insecurity of 802.11, in: *Proc. 7th ACM Conference on Mobile Computing and Networking (MOBICOM'01)*, Rome, Italy (2001).
- [6] N. Cam-Winget, R. Housley and J. Walker, Authenticated key exchange at the MAC layer (2001), Document number IEEE 802.11-01/573r0. Available from <http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/1-573.zip>.
- [7] Cisco Aironet security solution provides dynamic WEP to address researchers' concerns. Cisco Product Bulletin, No. 1281 (2001). http://www.cisco.com/warp/public/cc/pd/witc/ao/350ap/prodlit/1281_pp.htm.
- [8] Cisco wireless LAN security (2002). http://www.cisco.com/warp/public/779/smbiz/wireless/wlan_security.shtml/.
- [9] M.S. DeGraw-Bertsch, Configuring a FreeBSD access point for your wireless network, *Sys Admin* 11(3) (2002).
- [10] N. Ferguson, Michael: An improved MIC for 802.11 WEP (2002). Document number IEEE 802.11-02/020r0. Available from <http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/2-020.zip>.
- [11] S. Fluhrer, I. Mantin and A. Shamir, Weaknesses in the key scheduling algorithm of RC4, in: *Proc. 8th Workshop on Selected Areas in Cryptography, LNCS 2259*, Springer-Verlag (2001).
- [12] freeRADIUS (2003), v0.9.2. <http://www.freeradius.org/>.
- [13] P. Gutman, Cryptlib 3.0 (2002). Available from <http://www.cs.auckland.ac.nz/~pgut001/cryptlib/>.
- [14] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. ANSI/IEEE Standard 802.11 (1999).
- [15] Standards for local and metropolitan area networks: Standard for port based network access control. IEEE Draft P802.1X/D11 (2001).
- [16] Draft supplement to standard for telecommunications and information exchange between systems—LAN/MAN specific requirements—part 11: Wireless medium access control (MAC) and physical layer (PHY) specifications: Specification for enhanced security. Document number IEEE Std 802.11i/D3.0 (Nov. 2002).
- [17] Information technology—security techniques, key management, part 2: Mechanisms using symmetric techniques. ISO/IEC 11770-2, first edition (1996).
- [18] Linksys group—wireless (2002). <http://www.linksys.com/Products/>.
- [19] A. Mishra and W. A. Arbaugh, An initial security analysis of the IEEE 802.1x standard. Technical Report UMIACS-TR-2002-10, U. Maryland, College Park, MD 20742 (Feb. 2002).
- [20] J. Malinen, Personal communication (August 2002).
- [21] J. Malinen, Host AP driver for Intersil Prism2/2.5/3 (2002). <http://hostap.epitest.fi/>.
- [22] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, (CRC Press, Boca Raton, Florida, 1996).
- [23] ORiNOCO WEPplus whitepaper, Agere systems (2001). <http://www.orinocowireless.com/upload/documents/WEPplusWhitepaper.pdf>.
- [24] ORiNOCO wireless networks, Agere systems (2002). <http://www.orinocowireless.com/>.
- [25] Prism wireless LAN, Intersil. [http://www.intersil.com/design/prism/\(2002\)](http://www.intersil.com/design/prism/(2002)).
- [26] G. Rehm, 802.11b homebrew antenna shootout (February 2002). <http://www.turnpoint.net/wireless/has.html>.
- [27] R.L. Rivest, The RC4 encryption algorithm. RSA Data Security Inc. (proprietary) (March 1992).
- [28] N. Shankar, W.A. Arbaugh and K. Zhang, A transparent key management scheme for wireless LANs using DHCP. HP Labs Technical Report HPL-2001-227 (2001). <http://www.hpl.hp.com/techreports/2001/HPL-2001-227.html>.
- [29] B. Schneier, *Applied Cryptography*, 2nd edition (John Wiley & Sons, New York, 1996).
- [30] Secure hash standard, National Institute of Standards and Technology, NIST FIPS PUB 180-1, U.S. Department of Commerce (April 1995).
- [31] A. Stubblefield, J. Ioannidis and A. Rubin, Using the Fluhrer, Mantin, and Shamir attack to break WEP, in: *Proc. 9th Network and Distributed System Security Symposium (NDSS'02)*. The Internet Society (2002).
- [32] J. Tourrilhes, Wireless tools for Linux, Electronic publication, (2002) http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.
- [33] IEEE 802.11 wireless local area networks: The working group for WLAN (2002) standards, <http://grouper.ieee.org/groups/802/11/>.
- [34] A. Wool, Why security standards sometimes fail, *Communications of the ACM, Inside Risks Column* 45(12) (2002) 144.
- [35] A. Wool, A note on the fragility of the “Michael” message integrity code. *IEEE Trans. Wireless Communications* 3(5) (2004) 1459–1462.
- [36] A. Young and B. O'Hara, A re-key proposal, Document number IEEE 802.11-01/540r0 (2001). Available from <http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/1-540.zip>.



Avishai Wool received a B.Sc. (Cum Laude) in Mathematics and Computer Science from Tel Aviv University, Israel, in 1989. He received an M.Sc. and Ph.D. in Computer Science from the Weizmann Institute of Science, Israel, in 1992 and 1997, respectively. Dr. Wool then spent three years as a Member of Technical Staff at Bell Laboratories, Murray Hill, NJ, USA. In 2000 Dr. Wool co-founded Lumeta corporation, a startup company specializing in network security. Since 2002 Dr. Wool has been an Assistant

Professor at the Department of Electrical Engineering Systems, Tel Aviv University, Israel.

Dr. Wool is the creator of the Lumeta Firewall Analyzer. He is an associate editor of the ACM Transactions on Information and System Security. He has served on the program committee of the leading IEEE and ACM conferences on computer and network security. He is a member of the ACM, IEEE Computer Society, and USENIX. His research interests include firewall technology, network and wireless security, data communication networks, and distributed computing.

E-mail: yash@acm.org