IEEE P802.11
Wireless LANs

Unsafe at any key size; An analysis of the WEP encapsulation

Date: Oct 27, 2000

Author: Jesse R. Walker
Intel Corporation
2211 NE 25th Avenue
Hillsboro, Oregon 97124
Phone: +1 503 712 1849
Fax: +1 503 264 4843
e-Mail: jesse.walker@intel.com

Abstract

The IEEE 802.11 standard [1] defines the Wired Equivalent Privacy, or WEP, encapsulation of 802.11 data frames. The goal of WEP is to provide data privacy to the level of a wired network.

The 802.11 design community generally concedes that the WEP encapsulation fails to meet its design goal, but widely attributes this failure to WEP's use of 40-bit RC4 (see [2] or [3] for a description of RC4) as its encryption mechanism. Even at this late date, it is still repeatedly suggested, asserted, and assumed that WEP could meet its design goal by migrating from 40-bit to 104- or 128-bit RC4 keys instead.

This report seeks dispel this notion once and for all: it is infeasible to achieve privacy with the WEP encapsulation by simply increasing key size. The submission reports easily implemented, practical attacks against WEP that succeed regardless of the key size or the cipher. In particular, as currently defined, WEP's usage of encryption is a fundamentally unsound construction; the WEP encapsulation remains insecure whether its key length is 1 bit or 1000 or any other size whatsoever, and the same remains true when any other stream cipher replaces RC4. The weakness stems from WEP's usage of its initialization vector. This vulnerability prevents the WEP encapsulation from providing a meaningful notion of privacy at any key size.

The deficiency of the WEP encapsulation design arises from attempts to adapt RC4 to an environment for which it is poorly suited. This submission accordingly argues to replace RC4 by different cryptographic primitives in new work going forward. It identifies the characteristics needed by any encryption algorithm that can effectively provide data privacy in a wireless environment, and recommends candidate replacement algorithms and a replacement encapsulation.

1 Introduction

The IEEE 802.11 standard [1] defines the Wired Equivalent Privacy, or WEP, encapsulation of 802.11 data frames. The goal of WEP is to provide data privacy to the level of a wired network.

The 802.11 design community generally concedes that the WEP encapsulation fails to meet its design goal, but widely attributes this failure to WEP's use of 40-bit RC4 (see [2] or [3] for a description of RC4) as its encryption mechanism. Even at this late date, it is still repeatedly suggested, asserted, and assumed that WEP could meet its design goal by migrating from 40-bit to 104- or 128-bit RC4 keys instead.

This report seeks dispel this notion once and for all: it is infeasible to achieve privacy with the WEP encapsulation by simply increasing key size. The submission reports easily implemented, practical attacks against WEP that succeed regardless of the key size or the cipher. In particular, as currently defined, WEP's usage of encryption is a fundamentally unsound construction; the WEP encapsulation remains insecure whether its key length is 1 bit or 1000 or any other size whatsoever, and the same remains true when any other stream cipher replaces RC4. The weakness stems from WEP's usage of its initialization vector. This vulnerability prevents the WEP encapsulation from providing a meaningful notion of privacy at any key size.

The deficiency of the WEP encapsulation design arises from attempts to adapt RC4 to an environment for which it is poorly suited. This submission accordingly argues to replace RC4 by different cryptographic primitives in new work going forward. It identifies the characteristics needed by any encryption algorithm that can effectively provide data privacy in a wireless environment, and recommends candidate replacement algorithms and a replacement encapsulation.

This submission only discusses WEP's inability to correctly encrypt data. It is well known that encryption without data authentication is unsafe in a network. Very simple data modification attacks (e.g., modifications of DNS responses to redirect TCP/IP implementations to rogue web sites) exist against WEP, and these can be trivially implemented when one of the attacks enumerated below succeed against even a single packet. Therefore, any changes to the WEP encapsulation should also incorporate a keyed message authentication code to protect the encrypted data stream, even though this report doesn't focus on this aspect of WEP.

Little in this submission is original. Almost all of the reasoning already appears in some form in [2], [3], [4], and [5].

The remainder of this submission is organized as follows. Clause 2 reviews the WEP architecture. Clause 3 describes attacks to recover the key stream of any stream cipher as used by WEP, regardless of the cipher strength or key size. Clause 4 specializes the discussion to RC4 and identifies a number of problems making it unsuitable for the 802.11 environment. Finally Clause 5 summarizes the discussion by identifying more suitable cryptographic primitives and a better encapsulation scheme.

2 WEP overview

IEEE 802.11 defines a mechanism for encrypting the contents of 802.11 data frames. This scheme uses five elements directly relevant to its analysis:

a) A key shared between all the members of the BSS (there are really four shared keys, but this is irrelevant to the analysis).

b) An encryption algorithm. For WEP this is the RC4 stream cipher, used to generate a key stream, which is XORed against plaintext to produce ciphertext.

c) The corresponding decryption algorithm. For WEP this is the same as the encryption algorithm. RC4 is used to generate a key stream, which is XORed against the ciphertext to reproduce plaintext.

d) A 24-bit initialization vector, or IV. WEP appends the IV to the shared key; WEP uses this combined key and IV to generate the RC4 key schedule. WEP selects a new IV for every packet.

e) An encapsulation that transports the IV and the ciphertext from the sender (encryptor) to the receiver (decryptor).

f) WEP also uses a CRC of the frame payload plaintext in its encapsulation. The CRC is computed over the data payload and then appended to the payload before encryption. WEP encrypts the CRC with the rest of the data payload.

The operation of WEP is very simple to describe.

First, each member of the BSS is initialized with the shared key via an unspecified, implementation specific, out-of-band mechanism.

To send a WEP encapsulated frame, the sender calculates the CRC of the frame payload and appends it to the frame. It then selects a new IV, appends this to the shared key to form a "per-packet" key, and uses the result to generate an RC4 key schedule. The sender then uses RC4 to generate a key stream equal to the length of the frame payload plus CRC. The sender XORs the generated key stream against the plaintext payload data and CRC. The sender also inserts the IV into the appropriate field in the frame header, and sets a bit indicating this is a WEP encrypted packet. At this point, the WEP encapsulation is complete, and the frame can be sent to the peer.

To process a WEP frame, the receiver checks the "encrypted" bit in the arriving frame. If it is set, the receiver extracts the IV from the frame, appends it to the BSS shared key, and generates the "per-packet" RC4 key schedule. RC4 is applied to the key schedule to produce a key stream the length of the packet's encrypted payload. The receiver then XORs this key stream against the encrypted payload to extract plaintext. Finally the receiver verifies the CRC of the decrypted payload data to verify that the frame data correctly decrypted.

Several features of this design require comment.

The first is that the loss of a single bit of a data stream encrypted under RC4 causes the loss of all the data following the lost bit. This is because data loss desynchronizes the RC4 encryption and decryption engines. The resynchronization problem gets only worse as more bits become lost. Since 802.11 often drops entire packets, it infeasible to use a stream ciphers like RC4 across 802.11 frame boundaries. To be useful across packet boundaries, this environment instead requires that the cipher support a random access, "seek" type capability, where it is feasible to instantly and efficiently switch the cipher to any selected point in the key stream. Many stream ciphers offer random access support—any block cipher such as AES [8] in counter mode, for instance, or SEAL [9] for another. Instead of selecting a stream cipher with characteristics needed for a datagram environment, however, the WEP architecture accommodates itself to loss by reinitializing the cipher key schedule on every data frame.

Stream ciphers have a second property that is particularly important to the analysis: it is unsafe to use the same key twice, ever. Suppose the cipher produces a key stream of bits

$$k_1\, k_2\, k_3 \ldots$$

The encryptor uses the key stream sequence to encrypt the plaintext stream $p_1\, p_2\, p_3 \ldots$ into a ciphertext stream $c_1\, c_2\, c_3 \ldots$ by XORing each plaintext bit with the corresponding key stream bit:

$$c_i = p_i \oplus k_i, \text{ for } i = 1, 2, 3, \ldots$$

(Most practical implementations actually XOR whole bytes or words instead of bits.) The decryptor recovers the plaintext stream from the ciphertext stream by XORing each ciphertext bit with the corresponding key stream bit:

$$p_i = c_i \oplus k_i, \text{ for } i = 1, 2, 3, \ldots \text{ (*)}$$

The cipher stream is public knowledge, and it is presumed that adverasries will record the entire stream. If an adversary learns the plaintext value of bit i, she can recover the corresponding plaintext value of any other ciphertext stream encrypted from the same key stream: first compute the key stream bit

$$k_i = c_i \oplus p_i$$

and then use equation (*) to decrypt the corresponding bit of any other stream encrypted under the same key.

The WEP design attempts to accommodate this second property by introducing the IV. WEP combines the IV with the key to produce a new frame specific encryption key.

3 Decrypting data without keys

This clause begins by describing problems with the WEP IV. Then it turns to practical techniques to recover the WEP RC4 key stream, based on the WEP IV deficiencies. Finally it closes with a discussion of some issues around the analysis.

3.1 WEP IV problems

The WEP IV is 24 bits long. WEP appends the IV to the shared key to form a family of 2^{24} keys. As described above, each frame transmission selects one of these 2^{24} keys and encrypts the data under the key.

This scheme suffers from a basic problem. Since a stream cipher key stream can never be reused, it obliges the BSS to change the base key as soon as its members have consumed all of the 2^{24} keys derived from the base key. WEP defines no practical way to accomplish this, so in practice WEP keys are not replaced frequently enough to maintain the level of privacy intended. This leads to wide-spread key abuse; a single access point BSS running at 11 Mbps

and with a typical packet distribution can exhaust the derived key space in about an hour. A multi- access point network with tens or hundreds or thousands of access points can exhaust the key space at a faster rate, indeed, inversely proportional to the number of access points.

The problem is worse than this suggests, however. Since WEP shares the same base key among all the members of the BSS, and since the security of WEP depends on the <base-key, IV> pair never being recycled, WEP needs an IV avoidance algorithm, to prevent one node from reusing an IV already used by another. WEP defines no such algorithm, and it is unclear how to even begin to design one. A BSS could, for instance, partition the IV space among the BSS elements in a pre-defined manner, but this sort scheme either pre-supposes a static BSS membership static behavior, or some (secure) scheme to transfer an indication of which IVs have been used among members of the BSS, etc.

The usual way to avoid this kind of difficulty is to randomly select the IV instead. Random selection of the IV, however, presents its own difficulties because of the Birthday Paradox (see [2] or [3]). The Birthday Paradox is named for the counter-intuitive fact that, in a group of people as small as 23, there is a 50% chance that two members of the group will share the same birthday. In general, if a set has n members, and elements are selected from the set one at a time with replacement, then the probability of a duplicate after two draws is $p_2 = 1/n$ and, for $k \geq 3$, the probability of at least one duplicate is $p_k = p_{k-1} + (k-1) \cdot 1/n \cdot (1 - p_{k-1})$.

In the WEP case the IV space takes $n = 2^{24}$, and we exceed a 50% chance of a collision among IVs after only $k = 4823 \approx 2^{12}$ frames. The probability of collision is already 99% after 12,430 frames, or in 2 to 3 seconds of normal traffic at 11 Mbps. There is already a 10% chance of collision after 1881 frames, a 1% chance after 582, a 0.1% after 184 frames, 0.01% after 59, and 0.001% after only 19 frames. With randomly selected IVs, maintaining the five zeroes of assurance (0.000001%) becoming customary in many fields of computing requires changing the base key after all the members of a BSS have transmitted a total 6 frames under the key! The odds are a normal 11 Mbps BSS will begin to reuse keys in less than a second of operation, and there is a non-negligible probability that an attack can succeed well before this time has elapsed. The WEP IV space is far, far too small to protect against IV abuse.

It is important to be clear on what this does not say. It does not say that 50% of the IVs (and hence keys) will collide at about 2^{12} packets. It says that if an adversary collects a packet trace of about 2^{12} frames, there is about a 50% chance that the trace will contain at least one duplicate IV. But this is all the help an attacker needs.

3.2 Some attacks

So how does an attacker exploit the key reuse brought about by WEP's IV duplication? There is no magic here. All of the attacks are standard. WEP does not protect against any of them.

As with any stream cipher, an attacker can launch known plaintext and chosen ciphertext attacks. In today's computing environments, these attacks are ridiculously easy. The attacker Eve, for instance, can forge e-mail from Bob, asking the victim Alice to e-mail a file with known content. A telephone call from Eve works just as well, if Alice knows Eve or Eve can pass herself off as Bob. If Eve does not know Alice, she can use an e-mail anonymizer to send spam to her. All of this appears completely innocent, but it causes known text to be encrypted. Eve captures the encrypted text with a packet sniffer and uses the relation

$$k_i = c_i \oplus p_i$$

to recover the key stream. She can save any key streams generated this way, each indexed by the appropriate IV, and immediately decrypt any WEP frame she sees later (or appears earlier in the packet trace!) with the same IV.

Eve can work a little harder, with two sniffers, one on the radio link and one outside the firewall of the organization being attacked. Using a bit of traffic analysis, it is not difficult to correlate the unencrypted traffic recorded from outside the firewall with encrypted traffic captured from the over-the-air link. She repeats the same process above to recover more key streams.

When 802.11 is used as the data link for a TCP/IP network, every data frame contains an IP datagram conveying large amounts of known plaintext. In such an environment Eve can recover a partial key stream for *every* frame sent. Traffic analysis can usually identify the type of traffic fairly accurately even when it is encrypted, and this information can be used to guess the values of variable fields in the packet headers, such as IP addresses and protocol numbers. This reveals even more of the data and hence key stream. The known plaintext from a single DHCP exchange can provide sufficient information to decode almost the entire TCP/IP header of every subsequent IP datagram encrypted by the DHCP client. Similarly, any "decrypted" packet can provide context and hints to help identify the plaintext of a still not fully decrypted packet, and this process can continue until the key streams are revealed for almost every IV.

As easy as these attacks are, they are largely for amateurs; there is no need to work this hard; at 24 bits, collisions come to you rapidly, one on the order of every second or two. A serious attacker will simply trace the 802.11 frames and XOR together ciphertext streams produced under the same IV. If an IV is used at least twice, then the packet trace will show the cipher streams

$$c_i = p_i \oplus k_i, \text{ for } i = 1, 2, 3, \ldots$$
$$c_i' = p_i' \oplus k_i, \text{ for } i = 1, 2, 3, \ldots$$

produced from the key stream $k_1\ k_2\ k_3\ldots$ associated with the IV. XORing these together yields

$$p_i' \oplus p_i, \text{ for } i = 1, 2, 3, \ldots$$

i.e., the attacker knows with certainty the XOR of corresponding plaintext bits from $p_1'\ p_2'\ p_3'\ldots$ and $p_1\ p_2\ p_3\ldots$ This means the attacker can know when each bit is the same or different in the two streams, so can immediately reduce the number of possibilities for each byte pair <b, b′> from the two streams from 2^{16} possibilities to 2^8. Having a third or a fourth frame encrypted under the same IV can reduce the possibilities for the two plaintexts even more. These facts mean pattern recognition techniques can often split apart the intertwined plaintext streams.

For instance, in any computing system a considerable percentage of actual data exchanged over a link is ordinary text from some natural language. In any natural language represented by an alphabet, certain character sequences occur more frequently than others, and the probabilities for various character sequences have been computed from empirical measurement. These facts mean that with a fairly high probability the adversary can easily and mechanically eliminate all but one of the 2^8 possibilities for almost all of the byte pairs <b, b′>. The WEP checksum can be used to referee among guesses for the few that cannot be eliminated by probabilities. In this way the attacker can recover the plaintext and the key stream without knowing any of the plaintext in advance. There is a good reason why the operating instructions for RC4 explicitly warn never to use the same key twice!

As already noted, WEP as constituted today has no automated mechanism to change the BSS key, so people change it only infrequently, typically on the order of days or weeks or even months. After *n* hours of typical use, it is likely that the overwhelming number of IVs have been used at least *n*/2 times, so a packet trace of all the BSS traffic provides collisions among most of the IVs. In this way, an attacker can easily build up a dictionary of key streams for every IV. Since WEP frames are small, and since there are only 2^{24} possible IVs, an attacker using even obsolete hardware can afford to store the key streams for all the IVs associated with thousands of base keys. Given that people tend to cycle through a small number manually configured base keys, an attacker usually only has to resort to these techniques only a few times before the BSS's privacy is permanently compromised. It is simply not worth trying to provide any privacy if this is the best we can do.

There is still one more problem with WEP's use of the IV that is worth noting. As we have seen, it is feasible to recover the key streams associated with a large number of IVs in a short time. It is mechanical to use RC4's definition along with the key stream and plaintext to reconstruct the underlying key schedule. Not every recovered key stream will reconstruct the entire key schedule, but we can expect to recover a certain amount of these. When we have recovered a full key schedule, it is worth asking ourselves: can we also recover the key used to compute it? We know the IV corresponding to the key schedule is the least significant 24 bits of the key, and we know the deterministic algorithm generating key schedule from the key. Can we compute the key using this knowledge? Questions like this worry cryptographers. WEP's highly questionable method of concatenating the IV directly to the base key exposes the base key to direct attack. Because of this, it would also be advisable to find a new way to mix the IV with the base key.

3.3 Discussion

Attacks like these could be mounted against any protocol based on any stream cipher, but they are not, because they are ineffective in most environments. Why are they significant against WEP? why not against, e.g., SSL [7]? The reason is applications like SSL do not use RC4 the same way WEP uses it. SSL is designed to not reuse keys with any significant probability. In SSL, one random, e.g., 128-bit key is selected for an entire session. The key is not replaced on every packet. SSL can do this because it operates RC4 over a reliable data channel that does not lose data, so it can guarantee no synchronization loss between the encryptor and decryptor. When SSL replaces a key, it is not only the upper 24-bits that vary; it is the entire 128-bit key. This means the attacker has to collect samples from about 2^{64} SSL ciphertext streams rather than the 2^{12} needed to gain the same level of advantage in the case of WEP. That is, SSL's usage of the same encryption algorithm costs an attacker 2^{52}, or about 4,000,000,000,000,000, times more than WEP's.

SSL is too different from WEP for this comparison to be very meaningful. It is therefore instructive to examine how IPsec [6] avoids key collisions, as the encapsulation problem it faces is similar to 802.11's. IPsec uses four mechanisms:

1. It uses a unique key for each direction of each session over each (virtual) link. 3DES is the default cipher, and it has an effective key strength of 112 bits (the actual key size is 168 bits, but meet-in-the-middle attack against the 3DES construction diminishes the key strength). This means that there must be about $2^{56}/2 = 2^{55}$ sessions before there is any significant chance of collision between two randomly generated keys. An attacker has to assemble a ridiculously large database of ciphertexts before he has any hope of a discovering a collision. And 2^{55} requires about 2^{36} (64 billion) new sessions each second over twenty years, and the attacker somehow has to capture them all.
2. When using 3DES, each session takes its IVs from a 64-bit space (the IV comes from a 128-bit space when AES is used). This means there is no significant chance of a randomly selected IV collision until after about 2^{32} packets. By using such a large IV space, IPsec can avoid an impossible to define IV avoidance algorithm, relying instead on probability to make collisions too unlikely to worry about. The attacker, on the other hand, needs 2^{32} samples of ciphertext for each key in his database to make its existence worthwhile.
3. IPsec allows a single key to encrypt at most 2^{32} packets. The probabilities add up against an attack even if someone were able to somehow collect the necessary ciphertexts.
4. IPsec uses CBC mode. CBC mode uses the IV as a salt, not as part of the key. A salt is public information to help de-correlate the information encrypted under the key, so patterns in the plaintext become invisible in the ciphertext. And the way CBC mode works (XOR of the IV with the plaintext before encryption) makes IV reuse less catastrophic than in the WEP case.

Could the WEP encapsulation becomes safe if WEP migrates to per-link keys? Anyone who has followed the above analysis knows by now the answer is no, unless at a minimum WEP also defines an algorithm to make IV reuse a very rare event, and then also installs a new base key whenever the IV is in danger of repeating.

And even this “unless” is provisional at best, because presupposes there are no other fatal attacks against the WEP encapsulation. For instance, if each link has both a key for sending and receiving, then a revised WEP might try to define a deterministic algorithm to compute the next IV. For example, the algorithm might be to use a counter to represent the IV, or it might specify to execute the next step of a linear congruential pseudo-random number generator, allowing an implementation to avoid reuse of an IV until cycling through the entire space. But there is no evidence that such a scheme works from a security perspective, that the correctness of the encryption does not somehow depend on the randomness of the IV. Indeed, when the cipher is RC4, this determinism allows an attacker to launch weak key attacks on the cipher; see Section 4.

The problem is the existing WEP IV-based scheme is *ad hoc*, based purely on the notion that no attacks against it are evident to the designers, but, as we have seen, the scheme is fatally flawed in the light of its design goals. As a general rule, cryptographers distrust every *ad hoc* scheme until it is proven innocent, and that is the advice here: abandon the existing WEP scheme in favor of an algorithm known to provide privacy.

How a cipher is used is every bit as important as the strength of the cipher and its key length. WEP uses RC4 improperly, and because of this, fails to provide a meaningful notion of privacy. There are lots of ways to use a cipher securely, but WEP fails to make use of any of these proven techniques.

Another question that has been asked is whether the WEP encapsulation could be strengthened by including a *spoiler*. A spoiler is a random string inserted at the beginning of the plaintext data to be encrypted, with the intent of acting much like an IV, to de-correlate the encrypted text that follows. The answer here is also no, because the key stream generated by RC4 provides no feedback mechanism based on any previous plain or ciphertext, so the spoiler does not alter the key stream. Spoilers only provide security when the cipher is used in a feedback mode, as in CBC and OFB.

4 Problems with RC4

Most of the analysis thus far applies to any stream cipher, although we have noted a few problems with RC4 that makes its use by WEP a dubious decision, the most important being it has no random access capability. RC4 also suffers from one more problem that makes its applicability to 802.11 even more questionable.

One in every 256 RC4 key is “weak” [10], which means the key schedules for these keys are less correlated with the key than they ought to be. This makes it far easier to cryptanalyze data encrypted under these keys, in case we fix enough of the other WEP problems to justify its continued use in new submissions. If WEP were to continue to use RC4, there is a standard fix to this problem that should be incorporated into WEP’s definition. This is to run the RC4 key stream generator at least 256 steps each time its key schedule is re-initialized, i.e., to begin encrypting with the 257^{th} byte of the key stream instead of the first. Given that WEP already has to reinitialize the RC4 key schedule with every frame, this additional overhead is probably too much cost to bear.

One final problem should be noted but is not a concern. The key stream RC4 produces exhibits an empirically measurable bias, meaning it only imperfectly hides correlation in the encrypted data. This problem is inconsequential as long as WEP uses only an infinitesimal part of the generated key stream (there is not sufficient data to correlate, since WEP recomputes the key schedule on every frame) and far easier attacks against WEP's use of RC4 exist.

5 Summary and recommendations

This submission has identified significant deficiencies in the WEP data encapsulation that renders its data privacy claims meaningless, regardless of the key size. Increasing the WEP key from 40 to 104 or 128 bits does nothing to increase WEP's resistance to attack. This is because the deficiencies are related to how WEP uses cryptography, not the key size. WEP's design attempts to adapt RC4 to an environment for which it is poorly suited, with potentially catastrophic consequences for its intended users. Thus, to meet its goal of wired equivalent privacy, the WEP encapsulation needs significant reconstruction.

This section recommends changes to WEP to address these problems. It recommends a cipher, a mode of operation for the cipher, a key derivation algorithm (at least for manual keys), and random data recommendations. It concludes by recommending a new WEP encapsulation.

5.1 Cipher and mode of operation

All new symmetric key encryption efforts starting now should be based on the AES [8] block cipher. It is thought to be as good as any symmetric key cipher in the public domain, and allows for very efficient implementations over a very wide range of environments (8-bit processors to super computers). This submission recommends against RC4 for any future WEP encapsulation scheme; attempts to twist WEP around RC4 have led to enough problems already. The reformulated WEP should instead employ 128-bit AES as the mandatory to implement cipher.

This submission also recommends that WEP use AES in Offset Codebook Mode (OCB, [11]). This is a stream cipher that also produces a message authentication code, preventing an adversary from forging messages. The data encrypted under OCB is the same length as the plaintext data, a single key is used for both encryption and authentication, and it requires only the AES encryption, not decryption, engine. It is also a parallelizable mode, allowing for very high throughput. OCB has been optimized to minimize the number of calls to lower level cryptographic primitives, and can both encrypt/decrypt and tag/verify a message in a single pass.

The OCB state is the key, a stride, which provides the offset in the mode's name, and an IV. The stride and the IV are of the cipher block size (128 bits for AES). The stride is computed once per session. The per-frame overhead of OCB is 128 bits for the IV, and 128 bits for the OCB authentication tag.

5.2 Session key derivation

This submission recommends a session key derivation algorithm in the case of a manually configured base key, as used by WEP today. It does not recommend an algorithm for session key derivation when dynamic keying is available, because the scheme should incorporate state from the dynamic keying operation, to tie the key to the particular session that negotiated the key.

This algorithm produces two session keys, one for sending and the other for receiving.

1. Concatenate the (a) BSSID, (b) the sender's MAC address, and (c) the receiver's MAC address to produce a string. The order is important, as the two MAC addresses are reversed for sending and receiving.
2. Using the base key (manually configured key) and an IV of 128 zero bits, run the OCB-AES algorithm on the concatenated string. The session key is the authentication tag output by this:

$$\textit{session-key} \leftarrow \text{OCB-AES-tag}_{\text{base-key}}(0, \textit{BSSID} \mid \textit{sender-mac-addr} \mid \textit{receiver-mac-address})$$

Here '*a* | *b*' means the concatenation of strings *a* and *b*.

The motives for this algorithm are (a) to remove the base-key from direct attack and (b) weakly tie the session key to the particular parties using it. Under this algorithm different sets of peers use different session keys, even though all the members of the BSS share the same base key. Note that the keys produced by this algorithm are still subject to dictionary attack when the base key is a password or derived from a password by techniques such as PCKS #5 [12]. And all the keys are subject to spoofing if the base key is revealed to an adversary. There is no magic that can avoid these weaknesses.

5.3 Random data recommendations

The security of the recommended scheme depends critically on a source of cryptographically secure random numbers. Implementations must appear to produce random numbers for their nonces, even across power cycles and reboots. [13] and [14] discuss cryptographic requirements for randomness and provide useful implementation guidance.

5.4 Recommended encapsulation

The new WEP encapsulation this submission proposes consists of

1. A 128-bit IV. At session initiation the encrypter selects this randomly. On subsequent frames the encrypter may use the last 128-bits of encrypted data from the previous encrypted frame.
2. A 32-bit sequence number. This will be the first quantity encrypted. It indicates the number of frames sent under the present key. When manual keys are used, the sequence number is not used and set to 0. When dynamic keys are used, it is initialized to zero when the key is established.
3. The LLC data payload. This will also be encrypted.
4. The 128-bit OCB data authentication tag.

The encapsulation scheme thus adds 36 bytes to the frame size. This is the cost of privacy in a network these days.

To WEP encapsulate a frame, the encrypter first checks that the counter has not yet assumed the maximum value. If it has, the data stream encrypted under the present key terminates until the key management system replaces the base key. If the counter has not yet assumed its maximum value, and if dynamic keys are used, the encrypter increments the counter by 1 and inserts the new counter value into the appropriate location in the frame. The encrypter then selects the nonce and OCB encrypts/tags the counter and data. It inserts the OCG authentication tag at the tail of the frame. The frame is now ready to transmit.

To WEP decapsulate a received frame, the decryptor locates the correct context by the arriving MAC address. It uses the IV from the frame to OCB decypt the packet and verify its authenticity. If the session is dynamically keyed, it finally verifies that the counter is greater than any prior received value, i.e., the WEP receiver maintains a receive window size of 1. This prevents replay attack in the case when the session is dynamically keyed. Replay prevention is not feasible without dynamic keying

6 References

[1] IEEE Std 802.11-1997, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*

[2] B. Schneier, *Applied Cryptography*, 2nd Addition, Wiley, 1997.

[3] A. J. Menezes, P. C. van Orschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC, 1996

[4] S. M. Bellovin, "Problem Areas for IP Security Protocols", 6th USENIX UNIX Security Conference

[5] D. A. McGrew and S. R. Fluhrer, "The Stream Cipher Encapsulating Security Payload", draft-mcgrew-ipsec-scesp-01.txt, IETF, July 2000

[6] S. Kent and R. Atkinson, *RFC 2401, Security Architecture for the Internet Protocol*, IETF, November 1998

[7] T. Dierks and C. Allen, *RFC 2246, The TLS Protocol, Version 1.0*, IETF, January 1999

[8] J. Daemon and V. Rijmen, *AES Proposal: Rijndael*, available from http://csrc.nist.gov/encryption/aes/

[9] P. Rogaway and D. Coppersmith, "A Software-Oriented Encryption Algorithm", *in Fast Software Encryption, Cambridge Security Workshop*, Springer-Verlag, 1994

[10] A. Roos, *A Class of Weak Keys in the RC4 Stream Cipher*, attachment to e-mail to cipherpunks@toad.com, September 22, 1995

[11] P. Rogaway, *OCB Mode: Parallelizable Authenticated Encryption*, available from http://csrc.nist.gov/encryption/aes/

[12] *PKCS #5: Password-Based Encryption Standard*, RSA Laboratories, November 1993.

[13] D. Eastlake, S. Crocker, and J. Schiller, *RFC 1750, Randomness Recommendations For Security*, IETF, December 1994

[14] P. Gutman, *Random Number Generation*, available as http://www.cryptoengines.com/~peter/06_random.pdf