

802.11 Security Series

Part III: AES-based Encapsulations of 802.11 Data

Jesse Walker
Network Security Architect, Platform Networking Group
Intel Corporation

intel®

Agenda

This is the final of a series of three articles surveying security for IEEE 802.11 wireless LANs (WLANs) [1]. Part III examines the two *AES-based* encapsulation proposals. Since the 802.11 committee has not resolved whether to include only one or both proposals in the standard, this article will examine both. Neither AES-based protocol has a name, so we will follow customary practice and refer to each by the name of its mode of operation.

Review of Part I and II of this Series

[Part I](#) of this series reviewed the design flaws in WEP, or Wired Equivalency Protocol. The WEP design's major flaws are that it allows

1. Packet forgeries,
2. Replay attacks,
3. Weak-key attacks, and
4. Key-reuse attacks.
- 5.

These problems prevent WEP from meeting its goal of providing privacy for data transmitted over an 802.11 link.

[Part II](#) in our series studied TKIP (Temporal Key Integrity Protocol). TKIP was designed to “wrap around” WEP to fill the major gaps in the WEP design. TKIP defeats message forgeries by adding a *Message Integrity Code*, or *MIC*. It defeats replay attacks by enforcing *Initialization Vector*, or *IV*, sequencing. It defeats weak-key attacks by introducing a new per-packet key mixing function; and it defeats key reuse attacks by providing automatic key management. Thus, the TKIP extensions prevent all known WEP weaknesses.

However, as discussed in Part II, TKIP is not an ideal solution. The security assurances TKIP provides are still very weak. Security of the solution was compromised to permit the improved algorithms to be deployed as a software upgrade on existing hardware. TKIP is not viable as a long-term security solution because of its weakness. Since it is feasible to deploy more capable hardware in the long term, the 802.11 working group is designing more robust security protocols for the future.

Requirements

If one can design a new data encapsulation from scratch, it is not very difficult to specify the shape of the protocol, given the state of knowledge today. Many successful packet encapsulation schemes such as IPsec, SP3, and 802.10 have been developed over the past decade, so what is needed to provide privacy at the packet level is fairly well understood. Following are the basic requirements for such a protocol.

- It must use encryption properly.
- Encryption is not sufficient to meet the privacy requirements, however. The design must also defend against forgeries. It is infeasible to provide privacy in an environment that permits forgeries, because otherwise an attacker can use the protocol infrastructure itself to launch attacks.
- The design must defend against replays. A replay occurs when an attacker records and then later retransmits one or more packets. A replay attack is a special form of forgery that requires special techniques to defeat.
- In particular, a design must never reuse keys, because key reuse enables replay attacks. This is a more specific consequence of using encryption properly.
- The protocol must never reuse nonces or IVs or other information used to randomize the encryption function, as this constitutes a form of key reuse. This too is a more specific consequence of using encryption properly.
- It must protect the source and destination addresses from modification. The source address requires protection from modification to prevent identity hijacking. The destination address requires protection to prevent an attacker redirecting packets to an unauthorized receiver. This is a requirement unique to a hop-by-hop packet protocol.
- It should minimize the number of cryptographic primitives used, to minimize hardware costs, and to maximize usability by eliminating unnecessary deployment choices.
- It should minimize the software expenses. Access points will remain cost-critical, so the MIPS available for software implementations are expected to be minimal.
- It should use the best practice cryptographic primitives. This will maximize the lifetime of the security solution.

AES and Modes of Operation

The state-of-the-art bulk data encryption algorithm today is the *Advanced Encryption Standard*, or *AES*. AES became the Federal Government encryption standard in November 2001, defined in FIPS-197. It replaces the *Data Encryption Standard*, or *DES*, adopted in 1977.

AES is a *symmetric key block cipher*. A symmetric key cipher uses the same key for encryption and for decryption. AES supports 128-, 192-, and 256-bit keys. Longer keys imply more assurance when the cipher is properly used. Cryptographers believe that 128-bit keys will provide adequate security for any commercial or private application for many decades. Thus, AES's support for 192- and 256-bit keys is intended more as future proofing rather than as a practical necessity.

A block cipher operates on a byte string of a fixed size, whereas stream ciphers, like the RC4 cipher used in WEP, transform a single byte at a time. The number of bits in the block is called the cipher's *block size*. AES uses a block size of 128-bits, which is 16 bytes. If the data to be protected by a block cipher is not a multiple of the block size, then a padding scheme is required before application of the cipher to the data.

To use a block cipher one employs a *mode of operation*. A mode of operation is a precise recipe for using the cipher. Failure to follow the recipe can compromise the security guarantees of the block cipher. Some example modes of operation include *Electronic Codebook (ECB)* mode, *Counter (CTR)* mode, and *Cipher-Block Chaining (CBC)* mode.

ECB mode is a naïve use of a block cipher. To use the mode, partition a message M into blocks $M_1 M_2 \dots M_n$ and encrypt each one:

for $i = 1$ **to** n **do** $C_i \leftarrow E_K(M_i)$

where $E_K(\cdot)$ denotes encryption under the key K . The resulting sequence of blocks $C_1 C_2 \dots C_n$ is the encrypted message, where each block is simply the corresponding plaintext block encrypted under the key. Decryption reverses this process:

for $i = 1$ **to** n **do** $M_i \leftarrow D_K(C_i)$

where $D_K(\cdot)$ denotes decryption under the key K .

ECB mode is insecure for message streams that might repeat any data. The reason is it trivial to detect when two data blocks are the same. Indeed, suppose M and N are single block messages consisting of the same data. Then $E_K(M) = E_K(N)$, so an ECB based protocol publishes the when two blocks are the same or different, and this can be useful to an attacker.

Useful modes of operation leak information about the plaintext data much more slowly than does the ECB mode. Known useful modes leak information at the rate of $O(q^2/2^b)$, where q is the total number of blocks encrypted using the same key and b is the block size in bits. As an example, the old Data Encryption Standard takes $b = 64$ and AES takes $b = 128$. Thus, DES encryption loses all security after it has encrypted about $q = 2^{32}$ blocks under the same key, while AES methods are secure until about $q = 2^{64}$ blocks have been encrypted under the same key. Since the degradation is independent of the key size, the block size is a second parameter affecting security. One of the principle motivations for defining AES was that the 64-bit block size used by DES is too small to provide adequate security today.

Both the Counter and CBC mode leak information at the $O(q^2/2^b)$ rate, so both are practical ways to employ a block cipher. Both operate by employing auxiliary data to randomize the encryption operation, thereby preventing the sort of trivial information leakage that occurs with ECB mode.

Counter mode uses a monotonically increasing *counter* for the auxiliary data. If we again represent a message M as a partition of blocks $M_1 M_2 \dots M_n$, then counter mode can be described as

when the key K is assigned, set $counter \leftarrow 0$
 for each message $M = M_1 M_2 \dots M_n$
 $initial_counter \leftarrow counter$
 for $i = 1$ **to** n **do** $C_i \leftarrow M_i \oplus E_K(counter)$, $counter \leftarrow counter + 1$
 $encrypted_message = initial_counter C_1 C_2 \dots C_n$

That is, counter mode encrypts a *counter*, and increments the *counter* value by one for each block encrypted. The encrypted *counter* value is then XORed against the plaintext to produce ciphertext. The encryptor prepends the initial counter value to the encrypted ciphertext, to tell the decryptor where to restart the *counter*. The decryptor then extracts the plaintext thusly:

for an encrypted message $C = initial_counter C_1 C_2 \dots C_n$
 $counter \leftarrow initial_counter$
 for $i = 1$ **to** n **do** $M_i \leftarrow C_i \oplus E_K(counter)$, $counter \leftarrow counter + 1$
 $M = M_1 M_2 \dots M_n$

Notice that counter mode decryption only requires the block cipher encryption primitive, so it is relatively inexpensive to implement. Counter mode fails catastrophically if the counter value is ever repeated with the same key. In practice, this means it is unsafe to use counter mode without key management to provide a fresh key for every session.

CBC mode is the most widely deployed block cipher mode of operation. CBC mode uses a randomly selected *initialization vector*, or *IV*, to prevent trivial information leakage. CBC mode encryption of a message M may be summarized as

for each message $M = M_1 M_2 \dots M_n$
 $IV \leftarrow$ randomly selected value
 $initial_IV \leftarrow IV$
 for $i = 1$ **to** n **do** $C_i \leftarrow E_K(M_i \oplus IV)$, $IV \leftarrow C_i$
 $encrypted_message = initial_IV C_1 C_2 \dots C_n$

That is, CBC mode *whitens* the plaintext by XORing it with the randomly generated IV value with the plaintext prior to encryption. The encrypted XORed value becomes the next IV, to whiten the next plaintext. CBC mode repeats this process with each block of data in the

message. CBC mode selects a new random IV for each message, and fails catastrophically if the selected IV is not random. The encryptor prepends the initial IV value to the encrypted ciphertext, to tell the decryptor where to restart the *IV* value. The decryptor then extracts the plaintext by reversing the process:

```
for an encrypted message  $C = \text{initial-IV } C_1 C_2 \dots C_n$ 
   $IV \leftarrow \text{initial-IV}$ 
  for  $i = 1$  to  $n$  do  $M_i \leftarrow IV \oplus D_K(C_i), IV \leftarrow C_i$ 
 $M = M_1 M_2 \dots M_n$ 
```

In addition to providing data privacy, CBC mode can also be used to compute a *Message Integrity Code*, or *MIC*, to provide data authenticity. If *Known-IV* denotes some known IV value, then the CBC-MAC is defined as

```
for a message  $M = M_1 M_2 \dots M_n$ 
   $IV \leftarrow \text{Known-IV}$ 
  for  $i = 1$  to  $n$  do  $MIC_i \leftarrow E_K(M_i \oplus IV), IV \leftarrow MIC_i$ 
 $MIC = MIC_n;$ 
```

That is, the tag MIC_n becomes the MIC testifying to the authenticity of the data. If the *MIC* is sent with the message *M*, the authorized message receiver—i.e., the one holding the key *K*—can detect forgeries by recomputing the tag value over the received message. Just like CBC mode, the security of CBC-MAC decays at the rate $O(q^2/2^b)$, so when the underlying block cipher is AES, the block size $b = 128$, and it is safe to use CBC-MAC under the same key *K* for at most $q = 2^{64}$ blocks—a huge number that will never be attained by practical applications today.

Proposed 802.11 Encapsulations based on AES

IEEE 802.11 TG*i* (Task Group *i*) has received two proposals for a long-term encapsulation based on AES modes of operation. The first is called AES-CCM and the second AES-OCB.

The AES-CCM Encapsulation

The AES-CCM protocol is based on AES in Counter mode for data privacy and CBC-MAC for data authenticity. Development of the AES-CCM protocol has been attributed to Ferguson, Housley, Lennache, Stanley, and Whiting in [2] and [3].

The AES-CCM protocol requires two state variables. First, it employs a single-key AES key. The CCM protocol uses this key for both encryption and for computing a MIC. Using the same key for two different functions is normally considered a questionable practice. However, Jacob Jonsson of [RSA Labs](#) has proved that this is not a problem when the Counter mode counter and CBC-MAC IV are constructed as is done in the AES-CCM protocol.

The second state variable is a 48-bit packet sequence counter. The AES-CCM protocol uses the packet sequence counter to construct both the Counter mode encryption counter and the CBC-MAC IV. CCM constructs both the Counter mode counter and the CBC-MAC IV as the concatenation of the source MAC address, the packet sequence counter, a 16-bit zero per-packet block counter, and 16 bits of other data that distinguish the Counter mode counter from the CBC-MAC IV. The 16-bit per-packet block counter allows packet fragments (MPDUs or Message Protocol Data Units) consisting of 2^{16} blocks. Since each AES block consists of 128-bits = 16 bytes, this means the maximum amount of data the protocol supports is $2^{16} \cdot 16 = 2^{20} = 1048576$ bytes per packet fragment—far more “headroom” than the 2312-byte maximum supported by 802.11. The CCM protocol’s construction of the Counter mode encryption counter and the CBC-MAC IV from the packet sequence counter provides the key separation needed to use the same key for both encryption and the MIC.

The AES-CCM protocol encapsulates 802.11 MPDUs, or packet fragments, in four steps.

1. First, construct the Counter mode counter and CBC-MAC IV from the packet sequence counter, and then increment the counter by 1.
2. Once this is accomplished, use the AES key and the CMC-MAC IV to compute a MIC over the source and destination addresses, the QoS traffic class, the data length, and the MPDU data. Truncate the MIC value to 64-bits, and append the result to the MPDU data.
3. Use the AES key and the Counter mode counter to encrypt the MPDU data using AES Counter mode, including the appended MIC.
4. Complete the protected MPDU by inserting the packet sequence counter value between the 802.11 header and the encrypted data.

Thus, the AES-CCM protocol grows the MPDU data payload by 14 bytes—6 for the packet sequence counter value, and 8 for the encrypted MIC.

The AES-CCM protocol decapsulates a received MPDU by reversing these steps:

1. Extract the packet sequence counter from the received MPDU. If this value has been received already for the current AES key, discard the packet as a replay. Otherwise construct the Counter mode counter and CBC-MAC IV from the packet sequence counter.
2. Next, Counter-mode decrypt the encrypted payload, using the AES key and the constructed Counter-mode counter value.
3. Finally, recompute the MIC using the AES key and the CBC-MAC IV, truncate it to 64-bits, and compare the result against the decrypted MIC value in the received MPDU. If the two differ, discard the received packet fragment as a forgery; otherwise accept the received MPDU as genuine.

When used with a key management scheme, it is easy to see that this scheme meets the problem requirements. The MIC check makes forgeries computationally infeasible, and the packet sequence check prevents replays unless an attacker can create forgeries. The scheme never reuses a counter value or IV with the same key. The MIC protects the source and destination addresses against modification. It minimizes the number of cryptographic primitives to one—AES—and AES is the current best-practice cryptographic primitive. Furthermore, both Counter mode and CBC-MAC can be efficiently implemented in software.

The AES-OCB Encapsulation

The AES-OCB protocol is based on AES in **Offset Codebook** mode, abbreviated **OCB**. OCB is a novel mode of operation invented by Phil Rogaway, a cryptographer at the University of California at Davis. [4] describes OCB mode. The AES-OCB protocol is attributed to Cam-Winget and Walker in [5] and [6].

OCB mode provides both data privacy and data authenticity, using a single pass over the data and using a single key. The mode of operation gets its name from the value $O = E_K(0^b)$ — O denotes the b -bit zero vector encrypted under the key K , which is called an **offset**. The offset represents a “random” value that OCB mode uses to generate a sequence of values O_1, O_2, O_3, \dots that are at a maximal Hamming distance apart. OCB mode employs a **nonce** instead of an IV or counter to randomize the encryption. A nonce is any value that is used at most once in the context of a key. If N denotes the nonce for a message $M = M_1 M_2 \dots M_n$, OCB mode encrypts the i^{th} block M_i as $E_K(M_i \oplus O_i \oplus N) \oplus O_i \oplus N$; OCB mode encrypts the final message block in a slightly different way, as a value we will call Y_n , but the exact form is not important for understanding the operation. OCB mode computes a MIC as $E_K(M_1 \oplus \dots \oplus M_n \oplus Y_n \oplus O_{n+1} \oplus N)$.

Since OCB mode uses a single pass to both encrypt and authenticate the data, software implementations are about twice as fast as classical methods, such as those used by the AES-CCM protocol. What is even more interesting is that OCB mode has a proof of security giving precise bounds about information leakage. The OCB mode security theorem states that any attack against OCB mode can be turned into an attack to break the underlying cipher. Hence, if you believe a cipher like AES is secure, then using AES in OCB is also secure. The theorem computes that OCB mode leaks information concerning the protected data at a maximum rate of $3 \cdot q^2 / 2^{b+1}$ —here as before q is the total number of blocks encrypted under a single key, and b is the cipher block size.

Just like AES-CCM, the AES-OCB protocol requires two state variables. First, it employs a single-key AES key K . The AES-OCB protocol uses this key for both encapsulation and decapsulation.

The second state variable is a 28-bit packet sequence counter. The AES-OCB protocol uses the packet sequence counter to construct the OCB mode nonce. The protocol constructs the OCB mode nonce as the concatenation of the source and destination MAC addresses, the QoS traffic class, and the packet sequence counter.

The AES-OCB protocol operates on entire 802.11 packets (MSDUs), or entire packets. Encapsulation requires four steps.

1. First, construct the OCB mode nonce, and then increment the packet sequence counter by 1. Incrementing the packet sequence counter prevents future nonces from being reused with the same key.
2. Once this is done, use the AES key and nonce to OCB-mode encrypt the MSDU data, truncating the OCB mode MIC to 64-bits.
3. Complete the protected MSDU by inserting the packet sequence counter value between the MSDU header and its encrypted data.

The AES-OCB protocol grows the MSDU data payload by 12 bytes—4 for the packet sequence counter value, and 8 for the OCB MIC.

The AES-OCB protocol decapsulates a received MSDU by reversing these steps:

1. Extract the packet sequence counter from the received MSDU. If this value has been received already for the current AES key, discard the packet as a replay. Otherwise extract the source and destination addresses and QoS traffic class as well, and construct the OCB mode nonce.
2. Next, use the AES key to OCB-decrypt the MSDU data. If the decryption fails, the data is treated as a forgery; if it succeeds, accept the packet as genuine.

This scheme meets the problem requirements when used with a key management scheme. The nonce construction protects the source and destination addresses, the QoS traffic class, and the packet sequence number from modification or forgery. If an attacker attempts to alter any of these in a genuine packet, decryption fails at the receiver. The OCB MIC makes forgeries computationally infeasible, and the packet sequence check prevents replays unless an attacker could create forgeries. The scheme never reuses the packet sequence counter value with the same key. Therefore, the constructed nonces are all unique. It minimizes the number of cryptographic primitives to one, AES, the current best-practice cryptographic primitive. Lastly, OCB mode can be efficiently implemented in software, indeed more efficiently than conventional algorithms that make two passes over the packet data.

Comparison

Both of the AES-based protocols address all of the requirements. They differ on a number of points.

Data Authenticity

The proof of security for OCB mode demonstrates an exact upper bound on the advantage any possible attacker can achieve, $3 \cdot q^2 / 2^{b+1}$, where q is the total number of blocks encrypted under the same key, and b is the number of bits in the cipher block size. For AES, $b = 128$. With a sequence space of 2^{28} MSDUs and a maximum packet size of 2^{12} bytes = 2^8 blocks, the maximum number of blocks protected by a single key becomes $q = 2^8 \cdot 2^{28} = 2^{36}$, and the maximum advantage that any attacker can succeed in forging a packet is $3 \cdot (2^{36})^2 / 2^{129} = 3 \cdot 2^{72} / 2^{129} = 3 / 2^{57}$. This is not a significant threat.

The data authenticity of the AES-CCM protocol is not understood nearly as well. A proof yielding an $O(q^2 / 2^b)$ security bound exists for CBC-MAC as well, but it assumes that all messages have the same length. To compensate for this, the AES-CCM protocol protects the 802.11 length data length field by including this in the CBC-MAC computation and testing to see if the protected data actually consists of the number of bytes this field specifies. Cryptographers believe this yields a secure construction, but whether or not this is actually the case is unknown. Any mathematical proof showing this would be messy using known techniques. Thus, the OCB mode appears to be superior for data authentication.

Packet sequence space size

One of the motives for growing the AES-CCM protocol sequence space to 48-bits is to simplify 802.11 keying. The AES-OCB protocol needs to be revised to accommodate this simplification as well. However, doing so requires new techniques, since the current approach based on the nonce construction will not suffice. The nonce construction can be used to protect a maximum of 128-bits. The AES-OCB protocol's 28-bit packet sequence counter derives from the fact that this is all that remains from the nonce's 128-bits after including the source and destination addresses and the QoS traffic class.

A method for supporting a larger sequence space does exist, called the *associated data construction*. The associated data is "extra" data that must be protected from forgery but remain unencrypted. One can protect associated data by computing its CBC-MAC and XORing the result into the OCB MIC.

It is likely that the AES-OCB protocol will be adapted to use a 48-bit packet sequence space in the following way. The nonce will be constructed from the 48-bit packet sequence counter, the source address, and zero pad. The associated data will be constructed from the destination address, the QoS traffic class, and zero pad. The CBC-MAC of associated data will add one extra encryption per packet. The maximum advantage any possible attacker will have with this solution increases to $3 \cdot (2^{58})^2 / 2^{129} = 3 \cdot 2^{116} / 2^{129} = 3 / 2^{13}$. This is no longer an insignificant probability, but it is not a problem in practice. To attain this advantage, an attacker must choose all the data to be protected, and must also choose 2^{58} blocks in total.

MSDU v MPDU

The AES-CCM protocol protects MPDUs, the 802.11 packet fragments, while the AES-OCB protocol protects MSDUs, the 802.11 packets. Protecting MSDUs is more architecturally pure, meaning it leads to a more flexible and reusable implementation. Any protocol protecting entire MSDUs could be used with any 802 media. This approach also minimizes overhead. The AES-OCB protocol always adds 12 bytes of overhead to any packet. However, if an MSDU is fragmented into n MPDUs, the AES-CCM protocol adds $16 \cdot n$ bytes of overhead. In AES-CCM's favor, applying cryptographic protections to MPDUs is easier to do within hardware implementations of the 802.11 architecture.

Performance

There is no performance difference between the two approaches when both are implemented in hardware.

When software is used, the AES-OCB protocol enjoys about a 2:1 performance advantage over the AES-CCM protocol. This is due to the fact that AES-CCM requires twice as many cryptographic operations as AES-OCB, and cryptographic operations dominate all other costs in competent protocol implementations. AES can be implemented in software at a cost of about 20 to 30 cycles per byte. The maximum 802.11a user data rate is about 4.5 million bytes per second, and the maximum 802.11b user data rate is about 875 K bytes per second. This means that, for an 802.11 access point, the AES-OCB protocol requires dedication of about 90 to 135 MHz to encryption and AES-CCM requires about 180 to 270 MHz. For 802.11b access point, AES-OCB requires 17 to 26.5 MHz and AES-CCM requires 34 to 53 MHz.

Patent situation

The primary motive for the development of the AES-CCM protocol has been to avoid intellectual property claims. Three patents have been filed that might apply to OCB mode. Phil Rogaway, OCB mode's inventor, has filed for a patent covering the mode. Rogaway's starting point was a mode invented by Charanjit Jutla called IAPM, for which IBM has filed a patent. Finally, Virgil Gligor of the University of Maryland has invented a new mode of operation that uses similar techniques and has also filed a patent upon which OCB mode may infringe.

One faction within the 802.11 committee wants to avoid any patent-encumbered mechanisms entirely, and is fighting to insert AES-CCM into the security draft. This faction has sufficient votes to block the standard until AES-CCM is incorporated. On the other hand, a second faction of vendors has completed sufficient work on AES-OCB, and this faction has sufficient votes to prevent AES-OCB from being removed from the final standard. Thus, it appears that the only possible compromise may be to implement both.

Summary

IEEE 802.11 TGi has defined two AES-based protocols to provide long-term security for wireless LANs. Both proposals satisfy all of the requirements for such a protocol, and both will provide genuine security. The processing requirements for both protocols, however, will require the deployment of new access-point hardware. 802.11 will select one or both of these approaches for its next generation security needs.

For Further Reading

- [1] IEEE Std 802.11, Standards for Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.
- [2] Whiting, D., R. Housley, and N. Ferguson, "AES Encryption & Authentication using CTR Mode & CBC-MAC," IEEE 802.11 doc 02-001r1, March 5, 2002.
- [3] Letanche, O., and D. Stanley, "Proposed TGi D1.9 Clause 8 AES-CTR CBC-MAC (CCM) text," IEEE 802.11 doc 02-144r0, February 28, 2002.
- [4] Rogaway, P., M. Bellare, J. Black, and T. Krovetz, "OCB Mode," April 1, 2001, <http://www.cs.ucdavis.edu/~rogaway/ocb/ocb.htm>
- [5] Walker, J., "Unsafe at any key size: an analysis of the WEP encapsulation," IEEE 802.11 doc 00-362, October 27, 2000.
- [6] Cam-Winget, N., and J. Walker, "Motions to resolve comments on AES algorithms," IEEE 802.11 doc 01-382r0, July 9, 2001.

About the Author

Jesse Walker is the network security architect for Intel's Platform Networking Group. He is the technical editor for the 802.11 security enhancements, and was the first person to publicly identify the security flaws in the original 802.11 WLAN protocol. He joined Intel as part of the Shiva acquisition. Jesse has also been active in other industry standardization efforts, including IPsec. Jesse holds a Ph.D. in mathematics from the University of Texas at Austin. Jesse may be contacted at jesse.walker@intel.com.

