Submission to NIST:

Counter with CBC-MAC (CCM)

AES Mode of Operation

Submitter:

Russ Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
Phone: +1 703-435-1775
E-mail: RHousley@rsasecurity.com

Authors:

Doug Whiting
Hifn
5973 Avenida Encinas, #110
Carlsbad, CA 92009
Phone: +1 760-827-4502
E-mail: DWhiting@hifn.com

Russ Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
Phone: +1 703-435-1775
E-mail: RHousley@rsasecurity.com

Niels Ferguson
MacFergus BV
Bart de Ligtstraat 64
1097 JE Amsterdam, Netherlands
Phone: +31 20 463 0977
E-mail: Niels@ferguson.net

1. Mode Specification

CCM is a generic authenticate-and-encrypt block cipher mode. CCM is only defined for use with 128-bit block ciphers, such as AES. The CCM ideas can easily be extended to other block sizes, but this will require further definitions.

1.1. Generic CCM Mode

For the generic CCM mode there are two parameter choices. The first choice is *M*, the size of the authentication field. The choice of the value for M involves a trade-off between message expansion and the probability that an attacker can undetectably modify a message. Valid values are 4, 6, 8, 10, 12, 14, and 16 octets. The second choice is *L*, the size of the length field. This value requires a trade-off between the maximum message size and the size of the Nonce. Different applications require different trade-offs, so *L* is a parameter. Valid values of *L* range between 2octets and 8 octets (the value *L*=1 is reserved).

Name	Description	Field Size	Encoding of field
M	Number of octets in authentication field	3 bits	(*M*-2)/2
L	Number of octets in length field	3 bits	*L*-1

Parameters of CCM mode

1.2. Inputs

To send a message, the sender must provide the following information:

- An encryption key *K* suitable for the block cipher.
- A nonce *N* of 15-*L* octets. Within the scope of any encryption key *K*, the nonce value shall be unique. That is, the set of nonce values used with any given key shall not contain any duplicate values. Using the same nonce for two different messages encrypted with the same key destroys the security properties of this mode.
- The message *m*, consisting of a string of *l(m)* octets where $0 \le l(m) < 2^{8L}$. The length restriction ensures that *l(m)* can be encoded in a field of *L* octets.
- Additional authenticated data *a*, consisting of a string of *l(a)* octets where $0 \le l(a) < 2^{64}$. This additional data is authenticated but not encrypted, and is not included in the output of this mode. It can be used to authenticate plaintext packet headers, or contextual information that affects the interpretation of the message. Users who do not wish to authenticate additional data can provide a string of length zero.

Name	Description	Field Size	Encoding of field
K	Block cipher key	Depends on block cipher	String of octets.
N	Nonce	15-*L* octets	Not specified
m	Message to be encrypted and sent	*l(m)* octets	String of octets.
a	Additional authenticated data	*l(a)* octets	String of octets.

Inputs to CCM mode

1.3. Authentication

The first step is to compute the authentication field *T*. This is done using CBC-MAC. We first define a sequence of blocks B_0, B_1, ... B_n and then apply CBC-MAC to these blocks.

The first block B_0 is formatted as follows, where *l(m)* is encoded in most-significant-byte first order:

Octet no:	0	1 ... 15-*L*	16-*L* ... 15
Contents:	Flags	Nonce *N*	*l(m)*

Within the first block B_0, the Flags field is formatted as follows:

Bit no:	7	6	5	4	3	2	1	0
Contents:	Reserved	Adata	*M*			*L*		

The Reserved bit is reserved for future expansions and should always be set to zero. The Adata bit is set to zero if *l(a)*=0, and set to one if *l(a)*>0. The *M* field encodes the value of *M* as (*M*-2)/2. As *M* can take on the even values from 4 to 16, the 3-bit field can take on the values from 1 to 7. The *L* field encodes the size of the length field used to store *l(m)*. The parameter *L* can take on the values from 2 to 8 (recall, the value L=1 is reserved). This value is encoded in the 3-bit field using the values from 1 to 7 by choosing the field value as *L*-1 (the zero value is reserved).

If *l(a)*>0 (as indicated by the Adata field) then one or more blocks of authentication data are added. These blocks contain *l(a)* and *a* encoded in a reversible manner. We first construct a string that encodes *l(a)*.

If $0 < l(a) < 2^{16}-2^8$ then the length field is encoded as two octets which contain the value *l(a)* in most-significant-byte first order.

If $2^{16}-2^8 \le l(a) < 2^{32}$ then the length field is encoded as six octets consisting of the octets 0xff, 0xfe, and four octets encoding *l(a)* in most-significant-byte-first order.

If $2^{32} \le l(a) < 2^{64}$ then the length field is encoded as ten octets consisting of the octets 0xff, 0xff, and eight octets encoding *l(a)* in most-significant-byte-first order.

The length encoding conventions are summarized in the following table. Note that all fields are interpreted in most-significant-byte first order.

First two octets	Followed by	Comment
0x0000		Reserved
0x0001 ... 0xFEFF		For $0 < l(a) < 2^{16} - 2^8$
0xFF00 ... 0xFFFD		Reserved
0xFFFE	four octets *l(a)*	For $2^{16} - 2^8 \le l(a) < 2^{32}$
0xFFFF	eight octets *l(a)*	For $2^{32} \le l(a) < 2^{64}$

Length encoding for additional authenticated data

The blocks encoding a are formed by concatenating this string that encodes $l(a)$ with a itself, and splitting the result into 16-octet blocks, and then padding the last block with zeroes if necessary. These blocks are appended to the first block B_0.

After the (optional) additional authentication blocks have been added, we add the message blocks. The message blocks are formed by splitting the message m into 16-octet blocks, and then padding the last block with zeroes if necessary. If the message m consists of the empty string, then no blocks are added in this step.

The result is a sequence of blocks $B_0, B_1, ..., B_n$. The CBC-MAC is computed by:

$$X_1 := \mathrm{E}(K, B_0)$$

$$X_{i+1} := \mathrm{E}(K, X_i \oplus B_i) \quad \text{for } i=1, ... , n$$

$$T := \text{first-}M\text{-bytes}(X_{n+1})$$

where E() is the block cipher encryption function, and T is the MAC value. Note that the last block B_n is XORed with X_n and the result is encrypted with the block cipher. If needed, the ciphertext is truncated to give T.

1.4. Encryption

To encrypt the message data we use Counter (CTR) mode. We first define the key stream blocks by:

$$S_i := \mathrm{E}(K, A_i) \quad \text{for } i=0, 1, 2, ...$$

The values A_i are formatted as follows, where i is encoded in most-significant-byte first order:

Octet no:	0	1 ... 15-L	16-L ... 15
Contents:	Flags	Nonce N	Counter i

Within the each block A_i, the Flags field is formatted as follows:

Bit no:	7	6	5	4	3	2	1	0
Contents:	Reserved	Reserved	0			L		

The Reserved bits are reserved for future expansions and shall be set to zero. Bit 6 corresponds to the Adata bit in the B_0 block, but as this bit is not used here, it is reserved and shall be set to zero. Bits 3, 4, and 5 are also set to zero, ensuring that all the A blocks are distinct from B_0, which has the non-zero encoding of M in this position. Bits 0, 1, and 2 contain L, using the same encoding as in B_0.

The message is encrypted by XORing the octets of message m with the first $l(m)$ octets of the concatenation of $S_1, S_2, S_3, ...$. Note that S_0 is not used to encrypt the message.

The authentication value U is computed by encrypting T with the key stream block S_0 and truncating it to the desired length.

$$U := T \oplus \text{first-}M\text{-bytes}(S_0)$$

1.5. Output

The final result c consists of the encrypted message m, followed by the encrypted authentication value U.

1.6. Decryption and Authentication Checking

To decrypt a message the following information is required:

- The encryption key K.

- The nonce *N*.
- The additional authenticated data *a*.
- The encrypted and authenticated message *c*.

Decryption starts by recomputing the key stream to recover the message *m* and the MAC value *T*. The message and additional authentication data is then used to recompute the CBC-MAC value and check *T*.

If the *T* value is not correct, the receiver shall not reveal any information except for the fact that *T* is incorrect. The receiver shall not reveal the decrypted message, the value *T*, or any other information.

1.7. Restrictions

All implementations shall limit the total amount of data that is encrypted with a single key. The sender shall ensure that the total number of block cipher encryption operations in the CBC-MAC and encryption together shall not exceed 2^{61}. (This allows nearly 2^{64} octets to be encrypted and authenticated using CCM, which should be more than enough for most applications.) Receivers that do not expect to decrypt the same message twice may also check this limit.

The recipient shall verify the CBC-MAC before releasing any information such as the plaintext. If the CBC-MAC verification fails, the receiver shall destroy all information, except for the fact that the CBC-MAC verification failed.

1.8. Security Claim

We claim that this block cipher mode is secure against attackers limited to 2^{128} steps of operation if the key *K* is 256 bits or larger. There are fairly generic precomputation attacks against all block cipher modes that allow a meet-in-the-middle attack on the key *K*. If these attacks can be made, then the theoretical strength of this, and any other, block cipher mode is limited to $2^{n/2}$ where *n* is the number of bits in the key. The strength of the authentication is of course limited by *M*.

Users of smaller key sizes (e.g. 128-bits) should take precautions to make the precomputation attacks more difficult. Repeated use of the same nonce value (with different keys of course) must be avoided. One solution is to include a random value within the nonce. Of course, a packet counter is also needed within the nonce. Since the nonce is of limited size, a random value in the nonce provides a limited amount of additional security.

1.9. Security Proof

Jakob Jonsson from RSA Laboratories has developed a security proof of CCM. The resulting paper has been submitted for publication, so it will be available to everyone very soon. The proof shows that CCM provides a level of confidentiality and authenticity that is in line with other proposed authenticated encryption modes, such as OCB mode.

1.10. Rationale

The main difficulty in specifying this mode is the trade-off between nonce size and counter size. For a general mode we want to support large messages. Some applications use only small messages, but would rather have a larger nonce. Introducing the *L* parameter solves this issue. The parameter *M* gives the traditional trade-off between message expansion and probability of forgery. For most applications, we recommend choosing *M* at least 8.

The CBC-MAC is computed over a sequence of blocks that encode the relevant data in a unique way. Given the block sequence it is easy to recover *N*, *M*, *L*, *m*, and *a*. The length encoding of *a* was chosen to be simple and efficient when *a* is empty and when *a* is small. We expect that many implementations will limit the maximum size of *a*.

CCM encryption is a straightforward application of CTR mode. As some implementations will support a variable length counter field, we have ensured that the least significant octet of the counter is at one end of the field. This also ensures that the counter is aligned on the block boundary.

By encrypting T we avoid CBC-MAC collision attacks. If the block cipher behaves as a pseudo-random permutation, then the key stream is indistinguishable from a random string. Thus, the attacker gets no information about the CBC-MAC results. The only avenue of attack that is left is a differential-style attack, which has no significant chance of success if the block cipher is a pseudo-random permutation.

To simplify implementation we use the same block cipher key for the encryption and authentication functions. In our design this is not a problem. All the A blocks are different, and they are different from the B_0 block. If the block cipher behaves like a random permutation, then the outputs are independent of each other, up to the insignificant limitation that they are all different. The only places where the inputs to the block cipher can overlap is an overlap between an intermediate value in the CBC-MAC and one of the other encryptions. As all the intermediate values of the CBC-MAC computation are essentially random (because the block cipher behaves like a random permutation) the probability of such a collision is very small. Even if there is a collision, these values only affect T, which is encrypted so that an attacker cannot deduce any information, or detect any collision.

Care has been taken to ensure that the blocks used by the authentication function match up with the blocks used by the encryption function. This should simplify hardware implementations, and reduce the amount of byte-shifting required by software implementations.

1.11. Nonce Suggestions

The main requirement is that, within the scope of a single key, the nonce values are unique for each message. A common technique is to number messages sequentially, and to use this number as the nonce. Sequential message numbers are also used to detect replay attacks and to detect message reordering, so in many situations (e.g., IPsec ESP) the sequence numbers are already available.

Users of CCM, and all other block cipher modes, should be aware of precomputation attacks. These are effectively collision attacks on the cipher key. Let us suppose the key K is 128 bits, and the same nonce value N_0 is used with many different keys. The attacker chooses a particular nonce N_0. She chooses 2^{64} different keys at random and computes a table entry for each K value, generating a pair of the form (K,S_1). (Given the key and the nonce, computing S_1 is easy.) She then waits for messages to be sent with nonce N_0. We will assume the first 16 bytes of each message are known so that she can compute S_1 for each message. She looks in her table for a pair with a matching S_1 value. She can expect to find a match after checking about 2^{64} messages. Once a match is found, the other part of the matched pair is the key in question. The total workload of the attacker is only 2^{64} steps, rather than the expected 2^{128} steps. Similar precomputation attacks exist for all block cipher modes.

The main weapon against precomputation attacks is to use a larger key. Using a 256-bit key forces the attacker to perform at least 2^{128} precomputations, which is infeasible. In situations where using a large key is not possible or desirable (e.g., due to the resulting performance impact), users can use part of the nonce to reduce the number of times any specific nonce value is used with different keys. If there is room in the nonce, the sender could add a few random bytes, and send these random bytes along with the message. This makes the precomputation attack much harder, as the attacker now has to precompute a table for each of the possible random values. An alternative is to use something like the sender's Ethernet address. Note that due to the widespread use of DHCP and NAT, IP addresses are rarely unique. Including the Ethernet address forces the attacker to perform the precomputation specifically for a specific source address, and the resulting table could not be used to attack anyone else. Although these solutions can all work, they need careful analysis and almost never entirely prevent these attacks. Where possible, we recommend using a larger key, as this solves all the problems.

1.12. Efficiency

Encrypting and authenticating the empty message, without any additional authentication data, requires two block cipher encryption operations. For each block of additional authentication data one additional block

cipher encryption operation is required (if you include the length encoding). Each message block requires two block cipher encryption operations. The worst-case situation is when both the message and the additional authentication data are a single octet. In this case, CCM requires five block cipher encryption operations.

CCM results in the minimal possible message expansion; the only bits added are the authentication bits.

Both the CCM encryption and CCM decryption operations require only the block cipher encryption function. In AES, the encryption and decryption algorithms have some significant differences. Thus, using only the encrypt operation can lead to a significant savings in code size or hardware size.

In hardware, CCM can compute the message authentication code and perform encryption in a single pass. That is, the implementation does not have to complete calculation of the message authentication code before encryption can begin.

2. Summary of Properties

Security Function	authenticated encryption
Error Propagation	none
Synchronization	same nonce used by sender and recipient
Parallelizability	encryption can be parallelized, but authentication cannot
Keying Material Requirements	one key
Counter/IV/Nonce Requirements	counter and nonce are part of the counter block
Memory Requirements	requires memory for encrypt operation of the underlying block cipher, plaintext, ciphertext (expanded for CBC-MAC), and a per-packet counter (an integer; at most L octets in size)
Pre-processing Capability	encryption key stream can be precomputed, but authentication cannot
Message Length Requirements	octet aligned message of arbitrary length, up to 2^{8L} octets, and octet aligned arbitrary additional authenticated data, up to 2^{64} octets
Ciphertext Expansion	4, 6, 8, 10, 12, 14, or 16 octets depending on size of MAC selected

3. Test Vectors

The program used to generate test vectors is provided in CCM-TEST.ZIP. The expected output for the first test vector is provided here to allow implementers to verify that the test program is working.

```
=============== Packet Vector #1 ==================
AES Key:   C0 C1 C2 C3  C4 C5 C6 C7  C8 C9 CA CB  CC CD CE CF
      TA = A0 A1 A2 A3  A4 A5   48-bit pktNum = 0000.03020100
Total packet length =   31. [Input (8 cleartext header octets)]
           00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F
           10 11 12 13  14 15 16 17  18 19 1A 1B  1C 1D 1E
CBC IV in: 59 00 00 00  03 02 01 00  A0 A1 A2 A3  A4 A5 00 17
CBC IV out:EB 9D 55 47  73 09 55 AB  23 1E 0A 2D  FE 4B 90 D6
After xor: EB 95 55 46  71 0A 51 AE  25 19 0A 2D  FE 4B 90 D6   [hdr]
After AES: CD B6 41 1E  3C DC 9B 4F  5D 92 58 B6  9E E7 F0 91
After xor: C5 BF 4B 15  30 D1 95 40  4D 83 4A A5  8A F2 E6 86   [msg]
After AES: 9C 38 40 5E  A0 3C 1B C9  04 B5 8B 40  C7 6C A2 EB
After xor: 84 21 5A 45  BC 21 05 C9  04 B5 8B 40  C7 6C A2 EB   [msg]
After AES: 2D C6 97 E4  11 CA 83 A8  60 C2 C4 06  CC AA 54 2F
MIC tag  : 2D C6 97 E4  11 CA 83 A8
CTR Start: 01 00 00 00  03 02 01 00  A0 A1 A2 A3  A4 A5 00 01
CTR[0001]: 50 85 9D 91  6D CB 6D DD  E0 77 C2 D1  D4 EC 9F 97
CTR[0002]: 75 46 71 7A  C6 DE 9A FF  64 0C 9C 06  DE 6D 0D 8F
CTR[MIC ]: 3A 2E 46 C8  EC 33 A5 48
```

```
Total packet length =   39. [Encrypted]
           00 01 02 03  04 05 06 07  58 8C 97 9A  61 C6 63 D2
           F0 66 D0 C2  C0 F9 89 80  6D 5F 6B 61  DA C3 84 17
           E8 D1 2C FD  F9 26 E0
```

4. Performance Estimates

Performance depends on the speed of the block cipher implementation.

Encrypting and authenticating the empty message, without any additional authentication data, requires two block cipher encryption operations. For each block of additional authentication data one additional block cipher encryption operation is required (if you include the length encoding). Each message block requires two block cipher encryption operations. The worst-case situation is when both the message and the additional authentication data are a single octet. In this case, CCM requires five block cipher encryption operations. In hardware, for large packets, the speed achievable for CCM is roughly the same as that achievable with the CBC encryption mode.

5. Intellectual Property Statements

The authors hereby explicitly release any intellectual property rights to CCM to the public domain. Further, the authors are not aware of any patent or patent application anywhere in the world that covers CCM mode. It is our belief that CCM is a simple combination of well-established techniques, and we believe that CCM is obvious to a person of ordinary skill in the arts.