Project	IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)
Title	**Formal Specification of the CCM* Mode of Operation**
Date Submitted	September 9, 2005
Source	René Struik Certicom Corp. 5520 Explorer Drive, 4th Floor Mississauga, ON, Canada L4W 5L1 Voice: +1 (905) 501-6083 Fax: +1 (905) 507-4230 E-mail: rstruik@certicom.com
Re:	IEEE submission P802.15-02/469r0 (November 14, 2002), IEEE 802.15.4b/Draft D3.
Abstract	This document provides the formal specification of the CCM* mode of operation, as well as some (informational) design rationale. This document is an edited version of IEEE submission P802.15-02/469r0 by the same author and incorporates context on security and standardization efforts. This revision of 04/537r0 contains test vectors for 802.15.4b beacon, data, and command frames. (Note RS: This revision corrects some typographical errors in 04/537r1.)
Purpose	Facilitate adoption of the CCM* mode of operation as replacement of the security suites currently specified in the IEEE 802.15.4-2003 specification.
Notice	This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
Release	The contributor acknowledges and accepts that this contribution becomes the property of IEEE and may be made publicly available by P802.15.

Table of contents

1 Design rationale for the generic CCM* mode of operation

This note specifies CCM*, a block-cipher mode of operation that can be used to protect the privacy and/or authenticity of messages. This mode is a variant of the so-called CCM mode [4], aimed at broadening the applicability and usefulness hereof. First, we describe the CCM mode in some more detail and point out some of its properties. Second, we introduce the CCM* mode, a mode of operation that is interoperable with the CCM mode, while at the same time doing away with some of its limitations.

NIST Pub 800-38C [11] specifies the so-called CCM mode, a mode of operation that operates on block-ciphers with a 128-bit block size and involves a particular combination of the so-called Counter (CTR) mode of operation and the Cipher-Block Chaining (CBC) mode of operation [10], using a single key. The CCM mode of operation has acquired quite some practical relevance, due to the incorporation hereof as the mandatory block-cipher mode of operation in quite a few wireless standards that recently emerged, including the IEEE 802.11 WLAN standard [5] and the IEEE 802.15 High-Rate and Low-Rate WPAN standards [6], [7]. The CCM mode allows for variable-length authentication tags (from 32-bits to 128-bits), thus allowing varying degrees of protection against unauthorized modifications. The CCM mode allows quite efficient implementations, due to the fact that one only needs to implement the encryption transformation of the underlying block-cipher (and not the decryption transformation) and due to its reliance on a single key, rather than multiple keys, to provide confidentiality and authenticity services. This being said, the CCM mode has also some disadvantages, including the following:

- While the original CCM mode [4] provides for data authentication and, possibly, confidentiality, it does not provide for confidentiality only. This is unfortunate, since not all implementation environments call for data authenticity (e.g., when data authenticity is provided by an external mechanism). It would be advantageous to have a single mechanism that could provide for all suitable combinations of confidentiality and authenticity, rather than a strict subset hereof.
- The original CCM mode [4] is known to be vulnerable to specific attacks, if used with variable-length authentication tags rather than with fixed-length authentication tags only (see, e.g., Section 3.4 of [12]). Thus, the original CCM mode can only be securely used with the same key in settings with fixed-length authentication tags. This is unfortunate, since support for variable-length authentication tags is useful in constrained implementation environments, such as secured wireless sensor networks [7], where applications on a device might have different protection requirements, but would have to share the same key, due to resource constraints.

The CCM* mode extends the definition of the original CCM mode, such as to provide for confidentiality-only services, in addition to the other security service options already offered. Moreover, the CCM* mode adapts the original CCM mode such that the resulting mode can be used securely with variable-length authentication tags, rather than fixed-length authentication tags only. Thus, the CCM* mode takes away the disadvantages of the CCM mode pointed out above. For the detailed specification of the CCM* mode of operation, see Section 2.

At the same time, the specification of the CCM* mode is such that it is compatible with the CCM mode, as specified in the Draft Amendment (as of July 2003) to the IEEE 802.11 WLAN standard [5] and as specified in the IEEE 802.15.3 WPAN standard [6] and coincides with the specification of the default mode in the IEEE 802.15.4 WPAN standard [7].

The specification of the CCM* mode of operation imposes a specific choice of the input transformation (see Section 2.3.1.1), a specific representation of integers as octet strings, and a specific formatting of the *Flags* field used for authentication (see Section 2.3.1.2) and encryption (see Section 2.3.1.3). Obviously, variations hereof are possible, without necessarily impacting the security properties of the resulting mechanism (e.g., if the cryptographic assumptions discussed in Section 2.4 are adhered to). Obviously, other variations, such us relaxing restrictions on the nonce (see Section 2.3.1), are possible as well, possibly with an impact on the security properties of the resulting scheme.

2 Formal specification of the generic CCM* mode of operation

CCM* is a generic combined encryption and authentication block cipher mode. CCM* is only defined for use with block ciphers with a 128-bit block size, such as AES-128 [2]. The CCM* ideas can easily be extended to other block sizes, but this will require further definitions.

The CCM* mode coincides with the original CCM mode specification ([4], Appendix A of [10]) for messages that require authentication and, possibly, encryption, but does also offer support for messages that require only encryption. As with the CCM mode, the CCM* mode requires only one key. The security proof for the CCM mode [8], [9] carries over to the CCM* mode described here. The design of the CCM* mode takes into account the results of [12], thus allowing it to be securely used in implementation environments for which the use of variable-length authentication tags, rather than fixed-length authentication tags only, is beneficial.

2.1 Notation and representation

2.1.1 Strings and string operations

A string is a sequence of symbols over a specific set (e.g., the binary alphabet {0,1} or the set of all octets). The length of a string is the number of symbols it contains (over the same alphabet). The right-concatenation of two strings x and y (over the same alphabet) of length m and n respectively (notation: $x \| y$), is the string z of length $m+n$ that coincides with x on its leftmost m symbols and with y on its rightmost n symbols. An octet is a symbol string of length 8. In our context, all octets are strings over the binary alphabet.

2.1.2 Integers and their representation

Throughout this specification, the representation of integers as octet strings and of octets as binary strings shall be fixed. All integers shall be represented as octet strings in most-significant-octet first order. This representation conforms to the conventions in Section 4.3 of ANSI X9.63-2001 [1].

2.2 Symmetric-key cryptographic building blocks

The following symmetric-key cryptographic primitives and mechanisms are defined for use with all security-processing operations specified in this standard.

2.2.1 Block-cipher

The block-cipher used in this specification shall be the Advanced Encryption Standard AES-128, as specified in FIPS Pub 197 [2]. This block-cipher shall be used with symmetric keys with the same size as that of the block-cipher: 128 bits. These keys shall be generated uniformly at random. The procedure for generating keys is outside the scope of this standard.

2.2.2 Mode of operation

The block-cipher mode of operation used in this specification shall be the CCM* mode of operation, as specified in Section 2.3, with the following instantiations:

1. Each entity shall use the block-cipher E as specified in 2.2.1;
2. All octets shall be represented as binary strings in most-significant-bit first order;
3. The parameter L shall have the integer value 2;
4. The parameter M shall have one of the following integer values: 0, 4, 8, or 16.

2.3 Specification of CCM* mode of operation (in 'ANSI style')

Prerequisites: The following are the prerequisites for the operation of the generic CCM* mode:

1. A block-cipher encryption function *E* shall have been chosen, with a 128-bit block size. The length in bits of the keys used by the chosen encryption function is denoted by *keylen*.
2. A fixed representation of octets as binary strings shall have been chosen (e.g., most-significant-bit first order or least-significant-bit-first order).
3. The length *L* of the message length field, in octets, shall have been chosen. Valid values for *L* are the integers 2, 3, ..., 8 (the value *L*=1 is reserved).
4. The length *M* of the authentication field, in octets, shall have been chosen. Valid values for *M* are the integers 0, 4, 6, 8, 10, 12, 14, and 16. (The value *M*=0 corresponds to disabling authenticity, since then the authentication field is the empty string.)

2.3.1 CCM* mode encryption and authentication transformation

Input: The CCM* mode forward transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key sharing group member(s).
2. A nonce *N* of 15-*L* octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string *m* of length $l(m)$ octets, where $0 \leq l(m) < 2^{8L}$.
4. An octet string *a* of length $l(a)$ octets, where $0 \leq l(a) < 2^{64}$.

The nonce *N* shall encode the potential values for *M* such that one can uniquely determine from *N* the actually used value of *M*. The exact format of the nonce *N* is outside the scope of this specification and shall be determined and fixed by the actual implementation environment of the CCM* mode.

Note (informational): The exact format of the nonce *N* is left to the application, to allow simplified hardware and software implementations in particular settings. Actual implementations of the CCM* mode may restrict the values of *M* that are allowed throughout the life-cycle of the encryption key *Key* to a strict subset of those allowed in the generic CCM* mode. If so, the format of the nonce *N* shall be such that one can uniquely determine from *N* the actually used value of *M* in that particular subset. In particular, if *M* is fixed and the value *M*=0 is not allowed, then there are no restrictions on *N*, in which case the CCM* mode reduces to the CCM mode.

Actions: The CCM* mode forward transformation involves the execution, in order, of an input transformation (2.3.1.1), an authentication transformation (2.3.1.2), and an encryption transformation (2.3.1.3).

2.3.1.1 Input transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

This step involves the following steps, in order:

1. Form the octet string representation $L(a)$ of the length $l(a)$ of the octet string *a*, as follows:
   a. If $l(a)=0$, then $L(a)$ is the empty string.
   b. If $0 < l(a) < 2^{16}-2^{8}$, then $L(a)$ is the 2-octets encoding of $l(a)$.
   c. If $2^{16}-2^{8} \leq l(a) < 2^{32}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xfe, and the 4-octets encoding of $l(a)$.
   d. If $2^{32} \leq l(a) < 2^{64}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xff, and the 8-octets encoding of $l(a)$.

2. Right-concatenate the octet string *L(a)* with the octet string *a* itself. Note that the resulting string contains *l(a)* and *a* encoded in a reversible manner.

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.

4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16.

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   $$AuthData = AddAuthData \,||\, PlaintextData.$$

2.3.1.2 Authentication transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field consisting of the 1-bit *Reserved* field, the 1-bit *Adata* field, and the 3-bit representations of the integers *M* and *L*, as follows:

   $$Flags = Reserved \,||\, Adata \,||\, M \,||\, L.$$

   Here, the 1-bit *Reserved* field is reserved for future expansions and shall be set to ‘0’. The 1-bit *Adata* field is set to ‘0’ if $l(a)=0$, and set to ‘1’ if $l(a)>0$. The *M* field is the 3-bit representation of the integer $(M-2)/2$ if $M>0$ and of the integer 0 if $M=0$, in most-significant-bit-first order. The *L* field is the 3-bit representation of the integer $L-1$, in most-significant-bit-first order.

2. Form the 16-octet B_0 field consisting of the 1-octet *Flags* field defined above, the 15-*L* octet nonce field *N*, and the *L*-octet representation of the length field *l(m)*, as follows:

   $$B_0 = Flags \,||\, Nonce\ N \,||\, l(m).$$

3. Parse the message *AuthData* as $B_1 \,||\, B_2 \,||\, \ldots \,||B_t$, where each message block B_i is a 16-octet string.

4. The CBC-MAC value X_{t+1} is defined by

   $$X_0 := 0^{128};\ X_{i+1} := \mathrm{E}(Key, X_i \oplus B_i) \quad \text{for } i=0, \ldots, t.$$

   Here, $\mathrm{E}(K, x)$ is the cipher-text that results from encryption of the plaintext *x*, using the established block-cipher encryption function *E* with key *Key*; the string 0^{128} is the 16-octet all-zero bit string.

5. The authentication tag *T* is the result of omitting all but the leftmost *M* octets of the CBC-MAC value X_{t+1} thus computed.

2.3.1.3 Encryption transformation

The data *PlaintextData* that was established in clause 2.3.1.1 (step 4) and the authentication tag *T* that was established in clause 2.3.1.2 (step 5) shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet *Flags* field consisting of two 1-bit *Reserved* fields, and the 3-bit representations of the integers *0* and *L*, as follows:

   $$Flags = Reserved \,||\, Reserved \,||\, 0 \,||\, L.$$

Here, the two 1-bit *Reserved* fields are reserved for future expansions and shall be set to ‘0’. The *‘0’* field is the 3-bit representation of the integer 0, in most-significant-bit-first order. The *L* field is the 3-bit representation of the integer *L*-1, in most-significant-bit-first order.

2. Define the 16-octet A_i field consisting of the 1-octet *Flags* field defined above, the 15-*L* octet nonce field *N*, and the *L*-octet representation of the integer *i*, as follows:

   A_i = *Flags* || *Nonce N* || *Counter i*, for *i*=0, 1, 2, …

   Note that this definition ensures that all the A_i fields are distinct from the B_0 fields that are actually used, as those have a *Flags* field with a non-zero encoding of *M* in the positions where all A_i fields have an all-zero encoding of the integer 0 (see clause 2.3.1.2, step 2).

3. Parse the message *PlaintextData* as $M_1 || \ldots ||M_t$, where each message block M_i is a 16-octet string.

4. The ciphertext blocks $C_1, \ldots, C_t$ are defined by

   $C_i := E(Key, A_i) \oplus M_i$ for $i=1, 2, \ldots, t$.

5. The string *Ciphertext* is the result of omitting all but the leftmost *l(m)* octets of the string $C_1 || \ldots || C_t$.

6. Define the 16-octet encryption block S_0 by

   $S_0 := E(Key, A_0)$.

7. The encrypted authentication tag *U* is the result of XOR-ing the string consisting of the leftmost *M* octets of S_0 and the authentication tag *T*.

Output: If any of the above operations has failed, then output ‘invalid’. Otherwise, output the right-concatenation of the encrypted message *Ciphertext* and the encrypted authentication tag *U*.

2.3.2 CCM* mode decryption and authentication checking transformation

Input: The CCM* inverse transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key-sharing group member(s).
2. A nonce *N* of 15-*L* octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string *c* of length *l(c)* octets, where $0 \le l(c)-M < 2^{8L}$.
4. An octet string *a* of length *l(a)* octets, where $0 \le l(a) < 2^{64}$.

Actions: The CCM* mode inverse transformation involves the execution, in order, of a decryption transformation (2.3.2.1) and an authentication checking transformation (2.3.2.2).

2.3.2.1 Decryption transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the right-most string *U* is an *M*-octet string. If this operation fails, output ‘invalid’ and stop. *U* is the purported encrypted authentication tag. Note that the leftmost string *C* has length *l(c)-M* octets.

2. Form the padded message *CiphertextData* by right-concatenating the string C with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.
3. Use the encryption transformation in clause 2.3.1.3, with as inputs the data *CipherTextData* and the tag U.
4. Parse the output string resulting from applying this transformation as $m \| T$, where the right-most string T is an M-octet string. T is the purported authentication tag. Note that the leftmost string m has length $l(c)$-M octets.

2.3.2.2 Authentication checking transformation

The authentication checking transformation involves the following steps, in order:

1. Form the message *AuthData* using the input transformation in Clause 2.3.1.1, with as inputs the string a and the octet string m that was established in clause 2.3.2.1 (step 4).
2. Use the authentication transformation in Clause 2.3.1.2, with as input the message *AuthData*.
3. Compare the output tag *MACTag* resulting from this transformation with the tag T that was established in clause 2.3.2.1 (step 4). If $MACTag=T$, output 'valid'; otherwise, output 'invalid' and stop.

Output: If any of the above verifications has failed, then output 'invalid' and reject the octet string m. Otherwise, accept the octet string m and accept one of the key sharing group member(s) as the source of m.

2.3.3 Restrictions

All implementations shall limit the total amount of data that is encrypted with a single key. The CCM* encryption transformation shall invoke not more than 2^{61} block-cipher encryption function operations in total, both for the CBC-MAC and for the CTR encryption operations.

At CCM* decryption, one shall verify the (truncated) CBC-MAC before releasing any information, such as, e.g., plaintext. If the CBC-MAC verification fails, only the fact that the CBC-MAC verification failed shall be exposed; all other information shall be destroyed.

2.4 Security of CCM* mode of operation

The CCM* mode coincides with the original CCM mode specification [4] for messages that require authentication and, possibly, encryption, but also offers support for messages that require only encryption. As with the CCM mode, the CCM* mode requires only one key. The CCM* specification differs from the CCM specification, as follows:

- The CCM* mode allows the length of the authentication field M to be zero as well (the value M=0 corresponding to disabling authenticity, since then the authentication field is the empty string).
- The CCM* mode imposes a further restriction on the nonce N: it shall encode the potential values for M such that one can uniquely determine from N the actually used value of M.

As a result, if M is fixed and the value M=0 is not allowed, then there are no additional restrictions on N, in which case the CCM* mode reduces to the CCM mode. In particular, the proof of the CCM mode (see [8], [9]) applies.

For fixed-length authentication tags, the CCM* mode is equally secure as the original CCM mode. For variable-length authentication tags, the CCM* mode completely avoids – by design – the vulnerabilities that do apply to the original CCM mode.

For fixed-length authentication tags, the security proof of the original CCM mode carries over to that of the CCM* mode (also for $M=0$), by observing that the proof of the original CCM mode relies on the following properties, which slightly relax those stated in [8], [9] (relaxed property indicated in italics):

- The B_0 field uniquely determines the value of the nonce N.

- The authentication transformation operates on input strings $B_0 \| B_1 \| B_2 \| \ldots \|B_t$ from which one can uniquely determine the input strings *a* and *m* (as well as the nonce *N*). In fact, for any two input strings corresponding to distinct triples *(N, m, a)*, neither one is a prefix string of the other.
- All the A_i fields are distinct from the B_0 fields *that are actually used* (over the lifetime of the key), as those have a *Flags* field with a non-zero encoding of *M* in the positions where all A_i fields have an all-zero encoding of the integer 0.

Hence, if *M* is fixed, then the CCM* mode offers the same security properties as the original CCM mode: confidentiality over the input string *m* and data authenticity over the input strings *a* and *m*, relative to the length of the authentication tag. Obviously, if *M*=0, then no data authenticity is provided by the CCM* mode itself (but may be provided by an external mechanism).

For variable-length authentication tags, the original CCM mode is known to be vulnerable to specific attacks (see, e.g., Section 3.4 of [12]). These attacks may arise with the original CCM mode, since the decryption transformation does not depend on the length of the authentication tag itself. The CCM* mode avoids these attacks altogether, by requiring that one shall be able to uniquely determine the length of the applicable authentication tag from the A_i fields (i.e., from the counters blocks).

2.5 Interoperability between CCM mode and CCM* mode of operation

The CCM* mode reduces to the CCM mode in all implementation environments where the length of the authentication tag is fixed and where the value *M*=0 (encryption-only) is not allowed. In particular, the CCM* mode is compatible with the CCM mode, as specified in the Draft Amendment (as of July 2003) to the IEEE 802.11 WLAN standard [5] and as specified in the IEEE 802.15.3 WPAN standard [6]. The IEEE 802.15.4 WPAN standard [7] currently incorporates the CCM mode with variable-length authentication tags; the upcoming security amendment is anticipated to involve replacement of the CCM mode by the CCM* mode of operation, to securely support variable-length authentication tags in its target application area – low-cost sensor networks.

2.6 Test vectors for CCM* mode of operation – MAC Beacon Frame

The example below illustrates security processing of a beacon frame that is transmitted by the coordinator using its extended source address. In this example, the superframe specification field is set to 0xCF55 (e.g., *BO*=15) and there are no guaranteed time slots or pending addresses. This example uses source address 0xACDE480000000001, PAN identifier 0x4321, and beacon payload 0x51525354; the frame counter has integer value 5. The security level is set to 0x02 (MIC-64, or 64-bit data authenticity).

Unsecured beacon frame:

00 D0 84 21 43 01 00 00 00 00 48 DE AC || 55 CF 00 00 51 52 53 54.

Secured beacon frame:

08 D0 84 21 43 01 00 00 00 00 48 DE AC || 02 05 00 00 00 || 55 CF 00 00 51 52 53 54 22 3B C1 EC 84 1A B5 53.

2.6.1 CCM* mode encryption and authentication transformation

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce *N* of 15-*L*=13 octets to be used:

Nonce = AC DE 48 00 00 00 00 01 || 00 00 00 05 || 02.

3. The octet string *m* of length *l(m)*=0 octets to be used:

   m = ε (empty string).

4. The octet string *a* of length *l(a)*=26 octets to be used:

   a = 08 D0 84 21 43 || 01 00 00 00 00 48 DE AC || 02 || 05 00 00 00 || 55 CF 00 00 || 51 52 53 54.

Actions: The CCM* mode forward transformation involves the execution, in order, of an input transformation (2.6.1.1), an authentication transformation (2.6.1.2), and an encryption transformation (2.6.1.3).

2.6.1.1 Input transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*:

   L(a) = 00 1A.

2. Right-concatenate the octet string *L(a)* and the octet string *a* itself:

   L(a) || *a* =00 1A || 08 D0 84 21 43 01 00 00 00 00 48 DE AC 02 05 00 00 00 55 CF 00 00 51 52 53 54.

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.

   AddAuthData = 00 1A 08 D0 84 21 43 01 00 00 00 00 48 DE AC 02 05 00 00 00 55 CF 00 00 51 52 53 54 00 00 00 00.

4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16:

   PlaintextData = ε (empty string).

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   AuthData = 00 1A 08 D0 84 21 43 01 00 00 00 00 48 DE AC 02 05 00 00 00 55 CF 00 00 51 52 53 54 00 00 00 00.

2.6.1.2 Authentication transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field as follows:

   Flags = 59.

2. Form the 16-octet B_0 field as follows:

   B_0 = 59 || AC DE 48 00 00 00 00 01 00 00 00 05 02 || 00 00.

3. Parse the message *AuthData* as B_1 || B_2, where each message block B_i is a 16-octet string.

4. The CBC-MAC value X_3 is computed as follows:

i	B_i	X_i
0	59 AC DE 48 00 00 00 00 01 00 00 00 05 02 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1	00 1A 08 D0 84 21 43 01 00 00 00 00 48 DE AC 02	C4 A4 D0 BD 70 73 7E 32 11 2E 51 9A CA A2 01 F1
2	05 00 00 00 55 CF 00 00 51 52 53 54 00 00 00 00	A9 70 2C 6E E1 7E DE E0 C7 32 88 0A 40 41 7F 9C

3	—	AB 6B 19 E7 5B 75 2D 9A 6E F0 CC 13 09 98 EB D0

5. The authentication tag T is the result of omitting all but the leftmost M=8 octets of the CBC-MAC value X_3:

   T = AB 6B 19 E7 5B 75 2D 9A.

2.6.1.3 Encryption transformation

The data *PlaintextData* shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet Flags field as follows:

   Flags = 01.

2. Define the 16-octet A_i field as follows:

i	A_i
0	01 || AC DE 48 00 00 00 00 01 00 00 00 05 02 || 00 00

3. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$= 89 50 D8 0B DF 6F 98 C9 63 F2 D5 A1 08 A1 55 C7.

4. The encrypted authentication tag U is the result of XOR-ing the string consisting of the leftmost M=8 octets of S_0 and the authentication tag T:

   U = 22 3B C1 EC 84 1A B5 53.

Output: c=22 3B C1 EC 84 1A B5 53.

2.6.2 CCM* mode decryption and authentication checking transformation

Input: The inputs to the inverse mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce N of 15-L=13 octets to be used:

   Nonce = AC DE 48 00 00 00 00 01 || 00 00 00 05 || 02.

3. The octet string c of length $l(c)$=8 octets to be used:

   c = 22 3B C1 EC 84 1A B5 53.

4. The octet string a of length $l(a)$=26 octets to be used:

   a = 08 D0 84 21 43 || 01 00 00 00 00 48 DE AC || 02 || 05 00 00 00 || 55 CF 00 00 || 51 52 53 54.

Actions: The CCM* mode inverse transformation involves the execution, in order, of a decryption transformation (2.6.2.1) and an authentication checking transformation (2.6.2.2).

2.6.2.1 Decryption transformation

The decryption transformation involves the following steps, in order:

1. Parse the message c as $C \| U$, where the right-most string U is an M-octet string:

   $C = \varepsilon$ (empty string);
   U = 22 3B C1 EC 84 1A B5 53.

2. Form the 1-octet Flags field as follows:

   Flags = 01.

3. Define the 16-octet A_i field as follows:

i	$\boldsymbol{A_i}$
0	01 || AC DE 48 00 00 00 00 01 00 00 00 05 02 || 00 00

4. Define the 16-octet encryption block S_0 by

   S_0= $E(Key, A_0)$ = 89 50 D8 0B DF 6F 98 C9 63 F2 D5 A1 08 A1 55 C7.

5. The purported authentication tag *T* is the result of XOR-ing the string consisting of the leftmost *M=8* octets of S_0 and the octet string *U*:

   T = AB 6B 19 E7 5B 75 2D 9A.

2.6.2.2 Authentication checking transformation

The authentication checking transformation involves the following steps, in order:

1. Form the message *AuthData* using the input transformation in Clause 2.6.1.1, with as inputs the string *a* and the octet string *m*= ε (empty string):

   AuthData = 00 1A 08 D0 84 21 43 01 00 00 00 00 48 DE AC 02 05 00 00 00 55 CF 00 00 51 52 53 54
   00 00 00 00.

2. Use the authentication transformation in Clause 2.6.1.2, with as input the message *AuthData* to compute the authentication tag *MACTag*:

   MACTag = AB 6B 19 E7 5B 75 2D 9A.

3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in clause 2.6.2.1(step 5):

   T = AB 6B 19 E7 5B 75 2D 9A = *MACTag*.

Output: Since *MACTag*=*T*, output 'valid' and accept the octet strings *a* and *m* and accept one of the key sharing group member(s) as the source of *a* and *m*.

2.7 Test vectors for CCM* mode of operation – MAC Data Frame

The example below illustrates security processing of a data frame that is transmitted using extended addresses, with PAN identifier compression and acknowledgement enabled. This example uses source address 0xACDE0000000001, destination address 0xACDE480000000002, PAN identifier 0x4321, and data payload 0x61626364; the frame counter has integer value 5. The security level is set to 0x04 (ENC, or data confidentiality without data authenticity).

Unsecured data frame:

61 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC || 61 62 63 64.

Secured data frame:

69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC || 04 05 00 00 00 || D4 3E 02 2B.

2.7.1 CCM* mode encryption and authentication transformation

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 0.

Input: The inputs to the mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = AC DE 48 00 00 00 00 01 || 00 00 00 05 || 04.

3. The octet string *m* of length *l(m)*=4 octets to be used:

   m = 61 62 63 64.

4. The octet string *a* of length *l(a)*=26 octets to be used:

   a = 69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC || 04 || 05 00 00 00.

Actions: The CCM* mode forward transformation involves the execution, in order, of an input transformation (2.7.1.1), an authentication transformation (2.7.1.2), and an encryption transformation (2.7.1.3).

2.7.1.1 Input transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*:

   L(a) = 1A (the empty string). .

2. Right-concatenate the octet string *L(a)* and the octet string *a* itself:

   L(a) || *a* =00 1A ||69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC 04 05 00 00 00.

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.

   AddAuthData = 00 1A 69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC 04 05 00 00
   00 00 00 00 00.

4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16:

   PlaintextData = 61 62 63 64 00 00 00 00 00 00 00 00 00 00 00 00.

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   AuthData = 00 1A 69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC 04 05 00 00 00 00
   00 00 00 61 62 63 64 00 00 00 00 00 00 00 00 00 00 00 00.

2.7.1.2 Authentication transformation

T =ε (empty string).

2.7.1.3 Encryption transformation

The data *PlaintextData* shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet Flags field as follows:

   Flags = 01.

2. Define the 16-octet A_i field as follows:

i	A_i
1	01 || AC DE 48 00 00 00 00 01 00 00 00 05 04 || 00 01

3. Parse the message *PlaintextData* as M_1, where each message block M_i is a 16-octet string.
4. The ciphertext blocks C_1, C_2 are computed as follows:

i	***AES(Key,A$_i$)***	***C$_i$ = AES(Key,A$_i$) ⊕ M$_i$***
1	B5 5C 61 4F A6 8B 7E E0 CB 77 37 EB A8 1D 33 41	D4 3E 02 2B A6 8B 7E E0 CB 77 37 EB A8 1D 33 41

5. The string *Ciphertext* is the result of omitting all but the leftmost *l(m)*=4 octets of the string C_1:

   CipherText = D4 3E 02 2B.

Output: *c* = D4 3E 02 2B.

2.7.2 CCM* mode decryption and authentication checking transformation

Input: The inputs to the inverse mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:
   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.
2. The nonce *N* of 15-*L*=13 octets to be used:
   Nonce =   AC DE 48 00 00 00 00 01 || 00 00 00 05 || 04.
3. The octet string *c* of length *l(c)*=4 octets to be used:
   c = D4 3E 02 2B.
4. The octet string *a* of length *l(a)*=26 octets to be used:
   a = 69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC || 04 || 05 00 00 00.

Actions: The CCM* mode inverse transformation involves the execution, in order, of a decryption transformation (2.7.2.1) and an authentication checking transformation (2.7.2.2).

2.7.2.1 Decryption transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the right-most string *U* is an *M*-octet string:

   C = D4 3E 02 2B.

   U = ε (empty string).

2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.

   CipherTextData = D4 3E 02 2B 00 00 00 00 00 00 00 00 00 00 00 00.

3. Form the 1-octet Flags field as follows:

   Flags = 01.

4. Define the 16-octet A_i field as follows:

i	***A$_i$***
1	01 || AC DE 48 00 00 00 00 01 00 00 00 05 04 || 00 01

5. Parse the message *CiphertextData* as C_1, where each message block C_i is a 16-octet string.
6. The plaintext block P_1 is computed as follows:

i	***AES(Key,A$_i$)***	***P$_i$ = AES(Key,A$_i$) ⊕ C$_i$***

1	B5 5C 61 4F A6 8B 7E E0 CB 77 37 EB A8 1D 33 41	61 62 63 64 00 00 00 00 00 00 00 00 00 00 00 00

7. The octet string *m* is the result of omitting all but the leftmost *l(m)*=4 octets of the string P_1:

   m =61 62 63 64.

8. The purported authentication tag *T* is the empty string:

   T =ε (empty string).

2.7.2.2 Authentication checking transformation

The authentication checking transformation involves the following steps, in order:

1. Form the message *AuthData* using the input transformation in Clause 2.7.1.1, with as inputs the string *a* and the octet string *m* that was established in clause 2.7.2.1 (step 7):

   AuthData = 00 1A 69 DC 84 21 43 02 00 00 00 00 48 DE AC 01 00 00 00 00 48 DE AC 04 05 00 00 00 00 00 00 00 61 62 63 64 00 00 00 00 00 00 00 00 00 00 00 00.

2. Use the authentication transformation in Clause 2.7.1.2, with as input the message *AuthData* to compute the authentication tag *MACTag*:

   MACTag = ε (empty string).

3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in clause 2.7.2.1(step 8):

   T = ε = *MACTag*.

Output: Since *MACTag*=*T*, output 'valid' and accept the octet strings *a* and *m* and accept one of the key sharing group member(s) as the source of *a* and *m*.

2.8 Test vectors for CCM* mode of operation – MAC Command Frame

The example below illustrates security processing of an association request command frame that is transmitted by a FFD using extended addresses, with acknowledgement enabled. In this example, the capability field is set to 0xCE. This example uses source address 0xACDE480000000001, destination address 0xACDE480000000002, PAN identifier 0x4321, and command payload 0xCE; the frame counter has integer value 5. The security level is set to 0x06 (ENC-MIC-64, or data confidentiality with 64-bit data authenticity).

Unsecured command frame:

23 DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC || 01 CE.

Secured command frame:

2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC || 06 05 00 00 00 || 01 D8 4F DE 52 90 61 F9 C6 F1.

2.8.1 CCM* mode encryption and authentication transformation

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = AC DE 48 00 00 00 00 01 || 00 00 00 05 || 06.

3. The octet string *m* of length *l(m)*=1 octet to be used:

   m = CE.

4. The octet string *a* of length *l(a)*=29 octets to be used:

   a = 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF || 01 00 00 00 00 48 DE AC || 06 || 05 00 00 00 || 01.

Actions: The CCM* mode forward transformation involves the execution, in order, of an input transformation (2.8.1.1), an authentication transformation (2.8.1.2), and an encryption transformation (2.8.1.3).

2.8.1.1 Input transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*:

   L(a) = 00 1D.

2. Right-concatenate the octet string *L(a)* and the octet string *a* itself:

   L(a) || *a* = 00 1D || 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC 06 05 00 00 00 01.

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.

   AddAuthData = 00 1D 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC 06 05 00 00 00 01 00.

4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16:

   PlaintextData = CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   AuthData = 00 1D 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC 06 05 00 00 00 01 00 CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

2.8.1.2 Authentication transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field as follows:

   Flags = 59.

2. Form the 16-octet B_0 field as follows:

   B_0 = 59 || AC DE 48 00 00 00 00 01 00 00 00 05 06 || 00 01.

3. Parse the message *AuthData* as $B_1 \| B_2 \| B_3$, where each message block B_i is a 16-octet string.

4. The CBC-MAC value X_4 is computed as follows:

i	B_i	X_i
0	59 AC DE 48 00 00 00 00 01 00 00 00 05 06 00 01	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

1	00 1D 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF	1C E4 F7 E4 FC 48 74 6D 0C 22 20 5D E8 DB B9 B0
2	FF 01 00 00 00 00 48 DE AC 06 05 00 00 00 01 00	16 EC 61 6D 5A C1 1A A0 4B 30 89 09 5D D5 7F 89
3	CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	49 C3 1D 64 A5 A0 12 58 5B 07 78 B8 CD FE CE A8
4	—	D6 06 7B B5 9B 57 03 9C 00 98 0A 5D B3 63 BB 80

5. The authentication tag T is the result of omitting all but the leftmost M=8 octets of the CBC-MAC value X_4:

   T = D6 06 7B B5 9B 57 03 9C.

2.8.1.3 Encryption transformation

The data *PlaintextData* shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet Flags field as follows:

   Flags = 01.

2. Define the 16-octet A_i field as follows:

i	A_i
0	01 || AC DE 48 00 00 00 00 01 00 00 00 05 06 || 00 00
1	01 || AC DE 48 00 00 00 00 01 00 00 00 05 06 || 00 01

3. Parse the message *PlaintextData* as M_1, where each message block M_i is a 16-octet string.
4. The ciphertext block C_1 is computed as follows:

i	$AES(Key,A_i)$	$C_i = AES(Key,A_i) \oplus M_i$
1	16 A9 67 B4 0F F9 72 DE B1 CB 46 E7 09 FD EB FF	D8 A9 67 B4 0F F9 72 DE B1 CB 46 E7 09 FD EB FF

5. The string *Ciphertext* is the result of omitting all but the leftmost $l(m)$=1 octet of the string C_1:

   CipherText = D8.

6. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$= 99 D8 29 25 FA AE C5 6D 17 93 04 21 3B 88 69 35.

7. The encrypted authentication tag U is the result of XOR-ing the string consisting of the leftmost M=8 octets of S_0 and the authentication tag T:

   U = 4F DE 52 90 61 F9 C6 F1.

Output: c =D8 || 4F DE 52 90 61 F9 C6 F1.

2.8.2 CCM* mode decryption and authentication checking transformation

Input: The inputs to the inverse mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce N of 15-L=13 octets to be used:

   Nonce = AC DE 48 00 00 00 00 01 || 00 00 00 05 || 06.

3. The octet string c of length $l(c)$=9 octets to be used:

   c =D8 4F DE 52 90 61 F9 C6 F1.

4. The octet string a of length $l(a)$=29 octets to be used:

   a = 2B DC 84 21 43 02 00 00 00 00 48 DE AC FF FF || 01 00 00 00 00 48 DE AC || 06 || 05 00 00 00 || 01.

Actions: The CCM* mode inverse transformation involves the execution, in order, of a decryption transformation (2.8.2.1) and an authentication checking transformation (2.8.2.2).

2.8.2.1 Decryption transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the right-most string *U* is an *M*-octet string:

   C = D8;
   U =4F DE 52 90 61 F9 C6 F1.

2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.

   CipherTextData = D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

3. Form the 1-octet Flags field as follows:

   Flags = 01.

4. Define the 16-octet A_i field as follows:

i	A_i
0	01 || AC DE 48 00 00 00 00 01 00 00 00 05 06 || 00 00
1	01 || AC DE 48 00 00 00 00 01 00 00 00 05 06 || 00 01

5. Parse the message *CiphertextData* as C_1, where each message block C_i is a 16-octet string.

6. The plaintext block P_1 is computed as follows:

i	*AES(Key,A$_i$)*	$P_i = AES(Key,A_i) \oplus C_i$
1	16 A9 67 B4 0F F9 72 DE B1 CB 46 E7 09 FD EB FF	CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

7. The octet string *m* is the result of omitting all but the leftmost *l(m)*=1 octet of the string P_1:

   m = CE.

8. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$= 99 D8 29 25 FA AE C5 6D 17 93 04 21 3B 88 69 35.

9. The purported authentication tag *T* is the result of XOR-ing the string consisting of the leftmost *M=8* octets of S_0 and the octet string *U*:

   T = D6 06 7B B5 9B 57 03 9C.

2.8.2.2 Authentication checking transformation

The authentication checking transformation involves the following steps, in order:

1. Form the message *AuthData* using the input transformation in Clause 2.8.1.1, with as inputs the string *a* and the octet string *m* that was established in clause 2.8.2.1(step 7):

   AuthData = 00 1D DC 84 21 43 02 00 00 00 00 48 DE AC FF FF 01 00 00 00 00 48 DE AC 06 05 00 00
   00 01 00 CE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.

2. Use the authentication transformation in Clause 2.8.1.2, with as input the message *AuthData* to compute the authentication tag *MACTag*:

MACTag = D6 06 7B B5 9B 57 03 9C.

3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in clause 2.8.2.1(step 9):

   T = D6 06 7B B5 9B 57 03 9C = *MACTag*.

Output: Since *MACTag*=*T*, output 'valid' and accept the octet strings *a* and *m* and accept one of the key sharing group member(s) as the source of *a* and *m*.

References

[1] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography, American Bankers Association, November 20, 2001. Available from http://www.ansi.org.

[2] FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T, Springfield, Virginia, November 26, 2001. Available from http://csrc.nist.gov/.

[3] FIPS Pub 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198, US Department of Commerce/N.I.S.T., Springfield, Virginia, March 6, 2002. Available from http://csrc.nist.gov/.

[4] R. Housley, D. Whiting, N. Ferguson, Counter with CBC-MAC (CCM), submitted to N.I.S.T., June 3, 2002. Available from http://csrc.nist.gov/encryption/modes/proposedmodes/.

[5] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.11-1999, IEEE Standard for Telecommunications and Information Exchange Between Systems – LAN/MAN Specific Requirements – Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications, New York: IEEE Press, 1999.

[6] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.15.3-2003, IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPAN). New York: IEEE Press. 2003.

[7] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.15.4-2003, IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN). New York: IEEE Press. 2003.

[8] J. Jonsson, On the Security of CTR + CBC-MAC, in Proceedings of Selected Areas in Cryptography – SAC 2002, K. Nyberg, H. Heys, Eds., Lecture Notes in Computer Science, Vol. 2595, pp. 76-93, Berlin: Springer, 2002.

[9] J. Jonsson, On the Security of CTR + CBC-MAC, NIST Mode of Operation – Additional CCM Documentation. Available from http://csrc.nist.gov/encryption/modes/proposedmodes/.

[10] NIST Pub 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation – Methods and Techniques, NIST Special Publication 800-38A, 2001 Edition, US Department of Commerce/N.I.S.T., December 2001. Available from http://csrc.nist.gov/.

[11] NIST Pub 800-38C, Recommendation for Block Cipher Modes of Operation – The CCM Mode for Authentication and Confidentiality, NIST Special Publication 800-38C, US Department of Commerce/N.I.S.T., Springfield, Virginia, May 12, 2004. Available from http://csrc.nist.gov/.

[12] P. Rogaway, D. Wagner, A Critique of CCM, IACR ePrint Archive 2003-070, April 13, 2003.