Security mechanisms and vulnerabilities in the IEEE 802.15.3 wireless personal area networks

William Stewart

Department of Computer Science,
The University of Memphis,
Memphis, TN, USA
E-mail: will.stewart@medtronic.com

Yang Xiao*

Department of Computer Science,
The University of Alabama,
101 Houser Hall,
Tuscaloosa, AL 35487-0290 USA
E-mail: yangxiao@ieee.org
*Corresponding author

Bo Sun

Department of Computer Science,
Lamar University,
Beaumont, TX, USA
E-mail: bsun@cs.lamar.edu

Hsiao-Hwa Chen

Institute of Communications Engineering,
National Sun Yat-Sen University,
Kaohsiung City 804, Taiwan
E-mail: hshwchen@ieee.org

Abstract: The IEEE 802.15.3 specification is proposed for short-range high-speed Wireless Personal Area Networks (WPANs). Furthermore, it is proposed by the IEEE 802.15.3 a task group for Ultra-Wideband (UWB) transmission, which is an emerging wireless technology for future indoor and outdoor applications. This paper gives a survey on various security aspects and mechanisms in the IEEE 802.15.3 WPANs. It provides security analysis, points out some errors and security vulnerabilities of these networks and provides some corrections.

Keywords: IEEE 802.15.3; security; wireless personal area networks; WPANs.

Reference to this paper should be made as follows: Stewart, W., Xiao, Y., Sun, B. and Chen, H-H. (2007) 'Security mechanisms and vulnerabilities in the IEEE 802.15.3 wireless personal area networks', *Int. J. Wireless and Mobile Computing*, Vol. 2, No. 1, pp.14–27.

Biographical notes: William Stewart is currently an MS student in the Department of Computer Science at the University of Memphis. He received a BA in Communication at the University of Memphis in 2002. His research interests include compiler/interpreter design, cryptography, cryptanalysis, distributed systems and database systems.

Yang Xiao is currently in the Department of Computer Science at the University of Alabama. He was a voting member of IEEE 802.11 Working Group from 2001 to 2004. He is an IEEE Senior Member. He currently serves as Editor-in-Chief for *International Journal of Security and Networks (IJSN)*, *International Journal of Sensor Networks (IJSNet)* and *International Journal of Telemedicine and Applications (IJTA)*. His research areas are wireless networks, mobile computing, network security and telemedicine. He has published more than 190 papers in major journals (more than 50 in various IEEE Journals/magazines), refereed conference proceedings, book chapters related to these research areas.

Bo Sun received a PhD in Computer Science from Texas A&M University, College Station, USA, in 2004. He is now an Assistant Professor in the Department of Computer Science at Lamar University, USA. His research interests include the security issues (intrusion detection

in particular) of wireless ad hoc networks, wireless sensor networks, cellular mobile networks and other communications systems. He received the 2006 Texas Advanced Research Program award

Hsiao-Hwa Chen received a BSc and an MSc from Zhejiang University, China and a PhD from the University of Oulu, Finland, in 1982, 1985 and 1990, respectively, all in Electrical Engineering. He is a Full-time Professor and the Founding Director of the Institute of Communications Engineering of National Sun Yat-Sen University, Taiwan. He has authored or co-authored over 200 technical papers in major international journals and conferences, five books and several book chapters in the areas of communications. He served as a TPC Member on symposium co-chair for major international conferences, including IEEE VTC, IEEE ICC, IEEE Globecom, etc. He is serving as an Editor or Guest Editor for many major international journals.

1 Introduction

The IEEE 802.15.3 specification is proposed for high-speed short-range Wireless Personal Area Networks (WPANs) (IEEE, 2003). Aiming at low complexity, low cost and low power consumption, this specification specifies high data rate wireless communications among devices. It focuses on the power management, Quality of Service (QoS) capabilities and security issues in WPANs. Currently, the IEEE 802.15.3 Medium Access Control (MAC) protocol has been widely accepted by the industry, especially in the area of Ultra-Wideband (UWB) communications addressed by the IEEE 802.15.3 a task group.

This IEEE 802.15.3 specification primarily deals with small independent wireless devices that need to form special networks called *piconets*. A piconet is different from other types of data networks in which data communications are normally confined to a small area. Each piconet must have a Piconet Coordinator (PNC), which is generally a powerful device in the piconet. The PNC is responsible for managing the devices within the piconet so that resources can be efficiently utilised. The PNC provides timing information, management information, etc., for the piconet via beacon frames.

Because of unique security challenges faced by piconets, one focus of the specification is about how to provide security. Secure communications among devices are critical to ensure the security of the entire piconet. This includes security membership, key establishment, key transport, data confidentiality, integrity protection, etc. They are accomplished through suitable symmetric encryption, decryption and authentication mechanisms. For a piconet without implementing security, a device can join the piconet freely after its completion of an association process with the PNC. When security is enabled, that is, in the Mode 1 of the security specified by the IEEE 802.15.3 specification, the device must establish secure membership with the PNC before it can access the piconet's resources. This includes accessing the piconet's group data and communicating to the devices in the entire piconet.

This paper primarily focuses on the various security aspects of the 802.15.3 specification. Specifically, it includes encryption, decryption, key management, etc. We discuss the usage of the Advanced Encryption Specification (AES), the Cipher Block Chaining – Message Authentication Code (CBC-MAC) in the IEEE 802.15.3 specification, as well as key and group management issues. We assume that readers are already familiar with basic cryptographic algorithms (Stallings, 2003). This paper presents these security features in a more natural and concise manner than the overly complicated specification. Furthermore, we identify some vulnerabilities as well as errors in the IEEE 802.15.3 specification.

The rest of this paper is organised as follows. Section 2 introduces the IEEE 802.15.3 MAC protocol. Section 3 surveys security mechanisms of the IEEE 802.15.3 specification: cipher, authentication and encryption. We present secure piconet management and secure frame transmission in Sections 4 and 5, respectively. In Section 6, we provide a security analysis with an example. In Section 7, we identify some vulnerabilities and some errors, and we provide some corrections. Finally, we conclude this paper in Section 8.

2 IEEE 802.15.3 MAC

The 802.15.3 MAC (Chen et al., 2006; IEEE, 2003; Xiao, 2005; Xiao et al., accepted) adopts the notion of piconets. Each piconet has a master, called the piconet coordinator (PNC) and many slaves. Both the master and the slaves are devices (DEVs), while the master controls the slaves in its piconet. Peer-to-peer transmission can be provided with permission from the PNC. Connected to wireless or fixed links, the PNC assigns channel access times to the devices. In the 2.4–2.4835 GHz frequency range, four different channels can exist in a same area without interference to host four piconets and three different channels can exist in the same area without interference to host two piconets and an IEEE 802.11b Wireless Local Area Network (WLAN).

To join a piconet, a DEV first scans all the available channels and chooses one with an established piconet. Then, it sends an association request and in response, the PNC assigns the DEV a unique identifier, the DEVID, in the piconet. Channel time can be requested from the PNC, which makes decisions based on available resources in the piconet. A DEV can also request isochronous channel time if it needs channel time on a regular basis. Otherwise, it can make an asynchronous channel time request for a total amount of time to be used to transfer its data.

In the IEEE 802.15.3, the time is divided into *superframes*. The superframe is divided into three main parts: a beacon frame, the optional CAP and Channel Time Allocation Period (CTAP). The beacon frame is sent by the PNC to set the timing allocations and to broadcast management information for the piconet, including information for QoS management, power management and the basic timing. The CAP may be used for commands and non-stream data, as regulated by the PNC, while the CTAP is used for commands, isochronous streams and asynchronous data connections. During the CAP, the DEVs access the channel in a distributed manner using Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) and a backoff procedure. The PNC determines the length of the CAP and communicates with the DEVs in the piconet via the beacon frame. The CTAP adopts the TDMA protocol where the DEVs have specified time windows. The CTAP is composed of Channel Time Allocations (CTAs), including Management CTAs (MCTAs), that are used for communications between the DEVs and the PNC. The PNC controls the channel access by assigning CTAs to an individual DEV or to a group of DEVs with each CTA having a fixed start time and duration.

Four Interframe Spacings (IFSs) are defined in IEEE 802.15.3: the Minimum Interframe Space (MIFS), the Short Interframe Space (SIFS), the Backoff Interframe Space (BIFS) and the Retransmission Interframe Space (RIFS). The actual values of the MIFS, SIFS, BIFS and RIFS are physical layer (PHY) dependent.

IEEE 802.15.3 also defines three kinds of acknowledgement mechanisms: No-ACK (acknowledgement), Immediate-ACK and Delayed-ACK. For Immediate-ACK and Delayed-ACK, retransmission is adopted to recover corrupted frames in previous transmissions. For the No-ACK mechanism, ACK is not sent after a successful recipient. The no-ACK mechanism is useful for frames that do not require guaranteed delivery, where the retransmitted frame would arrive too late or an upper layer protocol is handling the ACK and retransmission protocol. In the Immediate-ACK mechanism, each frame is individually ACKed following the reception of the frame. The Delayed-ACK mechanism allows the source to send multiple frames without waiting for individual ACKs. Instead, the ACKs of the individual frames are grouped into a single response frame that is sent when requested by the source DEV.

Immediate-ACK frames and Delayed-ACK frames start transmission after a SIFS time at the end of the previous frame transmission. When either No-ACK or Delayed-ACK is used, an MIFS time is used in the CTA between a frame and the next successive frame transmitted over the medium.

On the other hand, during the CAP, CSMA/CA is used. The DEV or the PNC shall not transmit a frame if such a transmission as well as the ACK frame will go beyond the CAP. A DEV with a frame to transmit first senses the medium to become idle for a random length of time, called backoff time. Backoff is applied to every frame transmission during the CAP, except for the Immediate-ACK frame. In the CAP, the PNC can send a command after either a SIFS following the Immediate-ACK of a frame or following a frame with the ACK Policy field set to no-ACK. In this case, the PNC is not required to perform the backoff procedure before transmitting its frame.

3 Security mechanisms: cipher, authentication, encryption

In this section, we survey security mechanisms in the IEEE 802.15.3 specification. We first introduce the cipher used in the IEEE 802.15.3 specification, that is, the AES in Section 3.1. Then, we introduce how the authentication scheme and encryption scheme, that is, the CBC-MAC and CCM (counter mode encryption and CBC-MAC) used in the IEEE 802.15.3 specification in Sections 3.2 and 3.3, respectively.

3.1 Cipher

The AES is the US government formal specification based on Rijndael and it is a symmetric block cipher to replace its predecessor, the Data Encryption Specification (DES). Rijndael was named after the two Belgian crypographers: Joan Daemen and Vincent Rijmen. The IEEE 802.15.3 specification adopts the AES for the block cipher.

Rijndael supports a range of block and key sizes, whereas the AES adopts a 128-bit block size and a key size of 128, 192 or 256 bits. In the IEEE 802.15.3 specification, the key size is 128 bits. In the AES, a state is a 4 4 array of bytes and the AES operates on states. The AES includes 10 rounds, where each round includes four stages except the last round. The 128-bit (16 byte) block is depicted as a square matrix of 16 bytes. The block is copied into the state array. This state array is modified at each stage of encryption or decryption and copied into the output array at the end.

In each round of encryption and decryption, four different stages, one permutation and three substitution operations, are performed. They are: substitute bytes, shift rows, mix columns and add round key. In substitute bytes stage, an S-box is used to perform a byte-by-byte substitution of the block.

In the key expansion algorithm, the 128 bit key is taken as a square matrix of bytes. This key is then expanded into an array of 44 key scheduled words. Here the first four bytes of the 128 bits form the first column of the matrix, second 4 bytes form the second column and so on. The AES key encryption algorithm takes a 4 word key as input and gives a linear array of 44 words. Initially the 4 word key is copied into the first four words of the expanded key. Then the remainder of the expanded key is filled in four words at a time. Each word is obtained by XORing the values of immediately preceding word and the word four positions back. In case of the position which is a multiple of 4, function g is used, where the function g is composed of subfunctions such as Rot-word which performs a one-byte circular left shift on a word and subword which is used to have a byte substitution on each byte of its input word using the S-box. The above two subfunctions' results are XORed with a round constant. A round constant is a word in which three rightmost bytes are 0 so that the effect of an XOR with a round constant is performed on only the leftmost byte of the word.

The state array is subject to one permutation and three substitution operations in each round. The first one is the substitute bytes transformation. This substitute bytes transformation provides a non-linear operation to the algorithm. This step makes use of a 16 16 matrix of byte values called S-box. It contains a permutation of all possible 256 8-bit values. This is derived from the inverse function of the GF(2^8).

To avoid attacks, based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation, where affine is a transformation consisting of multiplication by a matrix followed by transformation the addition of a vector. This S-box also avoids fixed points and opposite fixed points. The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. By taking these row and column values as indices for the S-box, a unique 8-bit output value is chosen. In this way, each individual byte of the state is mapped into a new byte.

Shift row transformation is a transposition step where each row is shifted cyclically a particular number of times. This operates on the rows of the state and the first row of the state is not altered. The second row is altered by performing a 1-byte circular left shift. The third row is altered by a 2-bytes circular shift and the fourth row is altered by a 3-bytes circular shift. The inverse shift row transformation shifts the last three rows with a one byte circular right shift for the second row and so on in the opposite direction. Shift row transformations make sure that the 4 bytes of one column are spread out to four different columns.

A mix column transformation operates on each individual column. Here the 4 bytes of each column of the state array are combined using an invertible liner transformation. The liner transformation includes a product matrix obtained by applying individual additions and multiplications performed in GF(2^8). In the inverse mix column transformation, the result that the inverse transformation matrix times the forward transformation matrix equals the identity matrix. This mix columns transformation provides diffusion in the cipher.

The add round key transformation is used to combine the subkey with the state. In each round of the algorithm, 128 bits of the state are bitwise XORed with the 128 bits of the round key. This can be treated as the column-wise operation between the 4 bytes of the state column and one word of the round key. This operation affects each bit of state. Security is maintained by the complexity of the round key expansion and the complexity of the other stages of the AES.

The AES decryption cipher is not identical to the encryption cipher. Even though the form of key schedules for the encryption and the decryption are the same, the sequence of transformations for the decryption differs from that of the encryption.

3.2 Authentication

The IEEE 802.15.3 specification combines encryption (discussed in Section 3.3) and authentication in all of its secure messages with the exception of secure beacon frames, which only use an integrity code. For all non-beacon messages, an integrity code is generated, appended to the plaintext, that is, the payload (padded with zeros if needed) and then the two are encrypted together.

The integrity code for authentication is generated using a key, a nonce and the CBC-MAC. Additional authentication data and additional plaintext data can be used in the integrity code generation, but these are not required. The resulting integrity code should only be eight bytes in length. To authenticate the original message, the recipient needs to only generate the integrity code by the CBC-MAC based on the same data and then compare the received integrity code with the generated one. The message is authenticated if the two integrity codes match.

3.2.1 CBC-MAC generation

The CBC-MAC involves encrypting a block of plaintext data, taking the output block of ciphertext and XORing that block with the following block of plaintext before encrypting it. The nonce acts as the first block of data to be encrypted. After that, the optional padded authentication data is encrypted and then the optional padded plaintext data is encrypted. The first 8 bytes of the final block become the integrity code for the data.

Figure 1 illustrates how the CBC mode of encryption works (Stallings, 2003). This figure illustrates a typical process for encryption in this mode. In the IEEE 802.15.3 specification, this is used to generate the integrity code using the nonce, possibly some optional authentication data and possibly some optional plaintext data. Typically, the CBC mode of encryption will encrypt the first block with an Initial Value (IV) to make the encryption stronger and tougher to break. In the case of the IEEE 802.15.3 specification, instead of encrypting the first block of plaintext, the nonce is encrypted. One problem with this is that the nonce is 13 bytes long and the block size is 16 bytes long. The solution to this problem is that the IV is composed of the three sections shown in Table 1. The Section 1 is 1 byte long and is composed of $l(m)$, which is the length of the message m. The next 13 bytes are made up of the nonce and the last 2 bytes are made up of the authentication flags.

Figure 1 Cipher block chaining mode of encryption

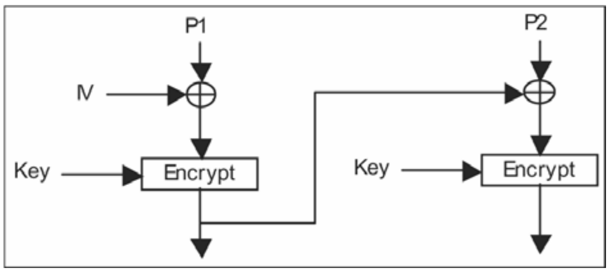

Table 1 Authentication block zero: the structure of B_0

2 bytes	*13 bytes*	*1 byte*
Authentication flags	nonce	length of *m*

3.2.2 Nonce

Table 2 shows that a nonce is made up of a total of 13 bytes, composed of a source ID (1 byte), a destination ID (1 byte), a time token (6 bytes), a secure frame counter (2 bytes) and a fragmentation control field (3 bytes) (IEEE, 2003). The key to prevent repetition is in the combination of the time token and the secure frame counter. The time token is unique to the superframe and the secure frame counter is incremented for each unique frame within the superframe. Of all these fields, the secure frame counter is the most critical for preventing the repetition of a nonce, at least while using the same key. The source and destination ID fields could obviously be repeated multiple times, because the source is likely to send multiple frames to the same destination throughout the lifetime of the piconet. The fragmentation control field could, under certain conditions, have a value of zero, so it offers no protection against repetition either.

Table 2 Nonce in the IEEE 802.15.3

3 bytes	*2 bytes*	*6 bytes*	*1 byte*	*1 byte*
Fragmentation control field	secure frame counter	time token	target ID	source ID

The time token and the secure frame counter are the only two fields that offer uniqueness for the nonce. The time token keeps track of which superframe the piconet is currently in. A superframe is like one round in a round robin task scheduling system. Within a single superframe, all the devices have the opportunity to send information and communicate multiple times. Due to the fact that multiple messages could be exchanged between two devices within the same superframe, the secure frame counter is incremented for each frame.

The only remaining loop-hole to the uniqueness of the nonce lies with the retransmission of messages. The 802.15.3 specification allows a message to be retransmitted with the same nonce if and only if the original message has not changed. If even one bit has changed, then the secure frame counter will need to be incremented using a new nonce and the message will need to be authenticated and encrypted again.

3.2.3 CBC-MAC decryption

Decrypting a secure message requires the key, the nonce, any additional authenticated data and the ciphertext. With these values, the ciphertext can be decrypted by simply using the key and the nonce in the Counter Mode encryption and then XORing the generated key stream with the ciphertext.

The counter is a combination of the nonce, some predefined flags and a counter starting at zero. Once again, the security of this method depends on the uniqueness of the nonce.

Once the ciphertext has been decrypted, the plaintext and the CBC-MAC are available to the device. The authentication step is performed by simply regenerating the CBC-MAC in the same way that the original integrity code is generated. If the two codes are identical, then the message is authenticated and passed on to the next layer. If the two codes are not identical, then the message is rejected and the only information that shall be made available is the fact that the message fails authentication. The rejected message is destroyed. Even a change in 1 bit, due to the avalanche effect, will cause the resulting integrity code to be quite different from the CBC-MAC that is appended to the original message.

3.2.4 Authentication flag and additional authentication data

The authentication data is optional for the CBC-MAC generation and whether it is used or not is specified in the authentication flags, shown in Table 4, in the first block.

The Adata field is set to 1 if the additional authentication data is going to be used. When decrypting this first block, if that bit is set to 1, then the first two bytes of the next block (B_1) help to determine the number of bytes used for $l(a)$, where a is additional data for authentication. If the first 2 bytes are between 0 0001 and 0 FEFF (in hexadecimal), then $l(a)$ is only using 2 bytes and the value in those bytes equals to the length of a in bytes. If the value of the first 2 bytes is 0 FFFE, then the next 4 bytes hold the value of $l(a)$. If the first 2 bytes equal to 0 FFFF, then the next 8 bytes hold the value of $l(a)$. Specifically, to ensure the correctness of the encoding, the first 2 bytes can contain only a value of up to 2^{16} 2^{8}. Any number higher than that can go into reserved codes for the first 2 bytes. As a result, if the number of bytes goes higher than 2^{16} 2^{8}, then the first 2 bytes are specially coded with 0 FFFE and then the following 4 bytes will hold the length of a. These 4 bytes allow for a to have up to 2^{32} 1 bytes in length. Finally, with a code of 0 FFFF, the following 8 bytes are used to show the size of a, allowing for up to 2^{64} 1 bytes.

The value L in Table 3 is to represent the number of bytes required to represent $l(m)$. Bits 0, 1 and 2 hold the value of L. The value M in Table 3 represents the number of bytes of the integrity code. Bits 3, 4 and 5 hold the value of (M 2)/2. The specification states that the integrity code should be 8 bits long, so that M should always be coded as eight and then (M 2)/2 6. The final bit in Table 3 is the reserved bit, which is always set to zero. At some point in the future, this bit might be used, but currently it is meaningless. Vendors are not even allowed to use this bit.

Table 3 Authentication flags

7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Reserved	Adata	*M*			*L*		

3.3 Encryption

For all non-beacon messages, an integrity code is generated, appended to the plaintext, that is, the payload and then the two are encrypted together.

The actual encryption of the plaintext along with the integrated code from the CBC-MAC is performed by using the AES in the CCM mode with a 128-bit key and a 128-bit block size. The counter encryption key stream is generated using the key and the nonce. The resulting key stream is XORed with the plaintext and the integrity code to produce the ciphertext. The integrity code is generated to allow the recipient to authenticate the plaintext.

3.3.1 CCM parameters and specification

In order to encrypt a message, the CCM function must have a key (K), a nonce (N), the message itself (m) and any additional data for authentication (a). The nonce is defined as being 15 L bytes long. Another parameter, M, indicates the number of bytes in the authentication field, which can have any even value from 4 to 16. The value of M is important, because it impacts how difficult it is to manipulate the data without being detected. The larger the M is, the more difficult the manipulation becomes. The specification has specified that M should equal to eight.

The first block is shown in Table 1. The string of blocks continues at B_1, shown in Table 4, with the blocks of the optional, additional authentication data (a). The length of the authentication data is put into the first block, followed by the authentication data itself. The remainder of the 16 byte block is padded with zeros for a total of B_k blocks. The last string of blocks contains the message itself, shown in Table 5, which contains blocks B_{k+1} to B_n.

Table 4 Additional authentication data (a) Blocks – the first 2, 6 or 10 bytes of B_1 show the length of the additional authentication data, which is optional for the authentication process. The remaining 14, 10 or 6 bytes of B_1 contain the beginning of a. The last block, B_k, is padded with zeros when necessary

16 L_2 bytes	*L_2 1 to 16*	*16 (k 2)*	*16 L_1*	*L_1 = 2, 6 or 10*
0	Remaining bytes of a	Middle bytes of a	First bytes of a	l(a)
B_k		B_{k-1} to B_2	B_1	

Table 5 Message Blocks (m) for Authentication – B_{k+1} to B_n contains the message padded with zeros in the final block (B_n). These blocks, combined with B_0 and B_1 through B_k, form the data for the CBC-MAC generation

16 L_3 bytes	*L_3 1 to 16*	*16 (n 2)*	*16*
0	Last bytes of m	Middle bytes of a	First bytes of m
B_n		B_{n-1} to B_{k+1}	B_{k+1}

Finally, the cipher block chaining algorithm is run over these blocks as follows:

$$X_0 = E\left(K, B_0\right) \tag{1}$$

$$X_{i+1} = E\left(K, B_{i+1} \oplus X_i\right) \tag{2}$$

$$T = \text{first } M \text{ octets}\left(X_{n+1}\right) \tag{3}$$

In this case, T is the integrity code that is used for message authentication. Due to the way that a CBC mode cipher propagates the influence of each bit throughout the entire encryption, this is a good method of authenticating the message, because any bit changes will yield a different integrity code for the recipient.

3.3.2 Counter mode encryption

The final encryption involves encrypting the plaintext and the integrity code with a counter mode encryption. The key stream is generated by simply encrypting the counter with the key (IEEE, 2003).

$$S_i = E\left(K, A_i\right) \tag{4}$$

The counter starts with A_0 as a block of data with the first byte being a counter (byte-0), the next 13 bytes being the nonce (bytes 1–13) and the last 2 bytes being encryption flags (bytes 14 and 15). These flags have a similar format to the flag bytes in the B_0 block. The first three bits hold the value of L, but the other entire bits equal to zero. Therefore, only bits 0, 1 and 2 may have ones in bytes 14 and 15.

The encryption flags are very similar to the authentication flags. The key difference is that the M field in the encryption fields is always zero.

The message is then encrypted, starting with block S_1. S_0 is reserved for the integrity code. Once the entire message has been encrypted, then the T value obtained from the last block of the CBC encryption is XORed with S_0 and truncated to only include M bytes. This becomes the integrity code that is added to the end of the frame data.

4 Secure piconet management

Secure piconet management is discussed in this section, including secure beacons, group key, group data key, etc.

4.1 Different keys

There are four different types of keys used in IEEE 802.15.3. Any secure frame will use either a management key or a data key. There are two different types of management keys and two types of data keys. The *PNC-Device management key* is used for communication between the PNC and one of the ACTIVE devices associated with the piconet. The *group data key* is mostly used for broadcasting within the group or when two devices do not have previously established peer management and data keys. There are two other keys, the *peer-to-peer management key (peer management key)* and the *peer-to-peer data key (peer data key)*, which are used for communication between two devices. The peer data key is only used for data transmission between devices. For most other communication, the peer management key is used. Overall, any device in a piconet must maintain a minimum of two keys: the PNC management key and the group data key. The device will need to store two more keys for each device to device secure relationship that it establishes after joining the piconet.

4.2 Secure beacon

A beacon provides essential information to the devices in a piconet. One essential part of the beacon is Secure Session Identifier (SECID) for the group key of the piconet. The SECID changes as the group key changes. The SECID is not the group key itself and it is only an identifier assigned to the group key.

The other essential part of the beacon that applies to secure piconets and secure superframes is the integrity protection. Beacons in secure piconets are protected by the group data key. Unlike most frames that are sent in a piconet, if the beacon fails its integrity check, then the device simply attempts to resynchronise its security state.

The beacon itself is not encrypted. An integrity code is added to the beacon for the purpose of ensuring that devices are synchronised with the PNC. Devices without the group data key ignore the SECID and secure frame counter until the group data key is obtained.

4.3 Changing the group key

When the PNC decides to change the group key, it must transmit the group key to each device in ACTIVE mode via a Distribute Key Request command. Simply distributing the key to the entire group using the current group key will disclose the key. One of the primary reasons for changing the group key is that a member of the piconet has left and additional communication cannot be counted as secure if former group members are able to decrypt communication. As a result, the Distribute Key Request command must be sent to each device which is still ACTIVE in the piconet using the PNC-Device management key. The new key is distributed with the corresponding SECID. Once all the devices have received the new group data key, then the PNC will begin broadcasting beacons with the new SECID. There is a limited window in which the previous group data key is still valid, but after a certain number of beacons, even the previous key will no longer be valid. While the PNC keeps a PNC-Device Management Key for each device in the piconet, the PNC is the only one guaranteed to have security relationships established with each device.

4.4 Changing piconet coordinators

When a piconet switches PNCs, the action is called the 'PNC Handover'. The current PNC sends a 'PNC Information' command to the future PNC along with a list of the devices in the piconet. At this point, the new PNC is in control of the piconet, but some commands may not be allowed until the new PNC has established secure PNC-Device management relationships (exchanged keys) with each device in the ad-hoc network. Since group membership has not changed, a new group key is not necessary.

4.5 When piconet membership status changes

There are a few cases when the piconet membership status changes. When a device joins or leaves the piconet, the membership status of the piconet has changed and the group data key must be changed. A key change can also count as a membership status change. There may be other situations that would require a piconet membership status change, but that would be caused by outside forces like the vendor's software.

If the target ID of a membership update message is the PNC ID, then the group data key or the PNC-device management key will apply to the message. If the target ID is not the PNC ID, then the peer-to-peer keys will be used. In addition to the target ID field, membership update messages also use a membership status field to indicate if the device currently has a secure relationship with the target device. The piconet membership status changes are managed by the MAC Layer Management Entity (MLME), which keeps track of the different security relationships among other things. The membership status is not in danger of being falsified, because this is generated by the device itself. These membership update messages are generated as a result of a Distribute Key message or some other events that would trigger a membership update message.

If the membership status field is set to MEMBER and the key info length field is not zero, then the key type identifies the type of key that will be updated or added. If the key info length field is set to zero, then the key type identifies the type of key that should be deleted. Once this key is deleted, the two devices, the target and the source, cannot transmit messages that require those keys.

4.6 The SECID field

The SECID field is used in the frame body of a secure frame to indicate the key to be used. This field is 2 bytes long composed of the Device ID in the lower byte and a unique identifier in the upper byte. The key originator proposes this unique identifier in a Distribute Key Message. The device ID for the group data key is equivalent to the broadcast identifier. Since the device ID is only 1 byte, there is a limit of 256 device IDs. In fact, there can only be 255 devices in a piconet, because one of the ID numbers is used for the broadcast ID. The purpose of the SECID is to identify the key with which the secure frame is encrypted. In a secure frame or distribute key message, the recipient will reject any SECID that does not have a matching device ID in the first byte.

4.7 When to use which keys

Not surprising is the fact that some devices may be associated with the piconet but not have a secure relationship established with all the other devices. Consequently, if a device wants to send a secure frame to another device, but the peer-to-peer data and management keys are not already known, then the device will attempt to obtain the keys. Devices that do not have a peer-to-peer relationship can use the group data key instead of the peer data key to transmit data frames. If the devices are unsuccessful in obtaining appropriate keys (i.e. they are not in the same piconet and do not have a secure relationship), then the devices simply will not send secure frames to each other.

In most instances for secure frames that are encrypted in IEEE 802.15.3, the peer management key will be used. When this is not available, meaning that an individual relationship does not exist between the two devices before this communication, then the group data key can be used instead. Data frames are the only exception to this rule, because they use the peer data key first before using the group data key as a last resort.

5 Secure frame transmission

5.1 Receiving secure non-beacon frames

When receiving a frame, a device must perform several checks before accepting the frame. The first check that is done before any other is the verification of the Frame Check Sequence (FCS). The FCS is a field at the end of the frame body that contains a Cyclic Redundancy Check (CRC). According to the specification, the calculation of the CRC is defined in ANSI X3.66-1979.

Once the FCS is validated, the device can check to see if the SECID is valid, meaning that the device ID built into the SECID identifies the correct source and the SECID is an acceptable type for the message type. This step can be skipped if the frame is not sent with security. If the SECID does not match any stored keys, then an UNAVAILABLE KEY error is generated and the frame is no longer processed. If the superframe is secure (i.e. the piconet is in security mode 1) and the device receives a non-secure command frame, then the device will reject the frame and generate an INVALID-SEC-VALUE security error.

Once the correct key has been selected, then the frame can be decrypted and authenticated. Note that the decryption must occur before the authentication, because the integrity code is appended to the end of the plaintext and encrypted at the same time. Once the plaintext and the integrity code have been acquired, then the device generates its own integrity code using CBC-MAC over the plaintext for authentication. If the integrity code included with the plaintext matches the integrity code generated by the device, then the message is authenticated. If the authentication fails, then a FAILED-SECURITY-CHECK security error is generated and the frame is no longer processed.

The above section identifies the steps for checking regular secure frames, but the secure beacon frames are little different and are discussed in the next subsection.

5.2 Receiving secure beacon frames

There is a large difference between secure non-beacon frames and secure beacon frames. When a device receives a secure beacon frame, it can be in two states. The first state in which the device is likely to be is in the role of a NON-MEMBER. When a device first decides to join a piconet, it does not yet have the group data key. At this point, the device does not know what the time token should be, nor does it have the means to find out. Once the group data key is obtained, then the device can determine the time token and becomes part of the group. The time token is very important for synchronisation within the piconet.

The device keeps track of two time tokens. One is the current time token, which is always equal to the time token in the most recent secure beacon. The other is the last valid time token, which is set to the last time token to arrive and be fully validated. If the SECID field does not match the piconet group data key stored in the device, then the device will need to request a new group data key. For either way, the current time token is incremented even when the last valid time token is not. If the group key is correct, then both time tokens are set to the time token in the secure beacon frame.

5.3 Integrity code generation and encryption for messages

There are five different kinds of secure messages that each use different methods of security by using different combinations of authentication data and message data. Table 6 shows the secure frame structure.

Table 6 Secure frame format – after the MAC header, every frame has the MAC frame body composed of the frame payload and the FCS. In a secure frame, the frame payload is further divided into the SECID, secure frame counter, the secure payload (encrypted message) and the integrity code

0 or 4 bytes	*8 bytes*	*Variable bytes*	*2 bytes*	*2 bytes*	*10 bytes*
FCS	Integrity code	Secure payload	Secure frame counter	SECID	MAC header
	Frame payload				

Note: The secure payload and the integrity code are encrypted.

To generate a *secure beacon integrity code*, the group data key is used as the key for this method and the entire beacon acts as the additional authentication data (*a*). The message data is an empty string, so there are no blocks of message data to include in the generation of the integrity code.

To generate a *secure command integrity code*, the integrity code is generated just like a secure beacon integrity code with the only difference being the key used.

For generating a *data integrity code*, the appropriate key is used. Once again, the circumstances under which the data is being transmitted will govern which key will be used, but generally, the peer data key will be used. Using the appropriate key, the integrity code is generated using the MAC header, SECID and secure frame counter as the authentication data and the actual data as the message. This is the first integrity code mentioned that includes the message text for the integrity code generation.

The *encrypted key message* is generated by using the appropriate management key for the actual encryption. The MAC header, SECID and secure frame counter fields are used as part of the authentication data for the CBC-MAC generation and the key is the message text. The key itself is preencrypted before being encrypted again by the CBC-MAC and CCM encryption methods.

For the generation of *encrypted data messages*, the same rules apply as the encrypted key message, except that the data key is used instead of the management key. The authentication data uses the frame data up to the message data and the message data is used as the message text input for the CCM operation.

6 Security analysis with an example

In this section, we put all of these steps together in an example. A device would first need to join a secure piconet. Unfortunately, joining a secure piconet is beyond the IEEE 802.15.3 specification.

6.1 Generating a beacon message

Every message starts with an MAC header, shown in Table 7. This contains the Frame Control field, the Piconet ID (PNID), the destination ID, the source ID, the fragmentation control field and the stream index. The source ID in the MAC header of a secure beacon message is going to be the PNC ID. The destination ID will be the broadcast ID, and therefore, all devices will listen to this message. This message also serves to synchronise the communication within the piconet.

Table 7 MAC header format – every frame that is sent out has this MAC header at the beginning of the message

1	*3*	*1*	*1*	*2*	*2*
Stream index	Fragmentation control	SrcID	DestID	PNID	Frame control
MAC header					

Once the piconet coordinator has generated the MAC header, the secure frame body must be generated, shown in Table 6. The first 2 bytes of the secure frame body are the SECID, which stipulates which key is being used for this secure message. In the case of a secure beacon message, the SECID will always indicate the current group data key.

Part of this beacon's purpose is to ensure that all devices have a current relationship with the piconet coordinator. If the device does not recognise the *SECID*, then the device knows that its relationship with the PNC is not current. The device will need to request the group data key from the PNC. For the purposes of this example, let us assume that the device has a current group data key.

After the *SECID*, the next necessary field for a secure beacon is the *Secure Frame Counter*. This is a critical aspect of the secure beacon, because the secure beacon holds a counter that increases with each superframe. This counter is critical to freshness protection and insuring that no device uses the same key and nonce combination. As a result, the secure frame counter field is a 2-bytes number that must increase with each secure superframe in the piconet.

After all these fields have been generated, the secure payload is added to the secure frame body. In the case of a secure beacon, the secure payload is made up of two sections: the synchronisation parameters and the information elements.

6.2 Why the CBC-MAC is secure

The integrity code is generated by the CBC-MAC to provide a means of authenticating the received message. The critical questions are

1 how this code assures a device in the piconet that the received message is from the one who claims to have sent it and

2 how this code assures the device that the message is not manipulated or tampered with during transmission.

Basically, the integrity code is the first 8 bytes of the last block from a cipher block chaining encryption algorithm using AES. In Layman's terms, the AES algorithm is run over a certain set of data in cipher block chaining mode. In this example, the data is the secure beacon itself. Cipher block chaining mode takes the output of one block's encryption and XORs that with the next block of plaintext. The result of the XOR operation is then encrypted using AES and the key, and the result is XORed with the following block. This mode of encryption is described in more detail above. The algorithm continues in a similar fashion until it reaches the last block of plaintext.

Part of the result of the last block's encryption is then used as the integrity code and the specification stipulates the first 8 bytes of this final block of ciphertext to be used as the integrity code for this beacon. One might think that this would make the message half as secure. However, because of the nature of AES and the CBC mode of encryption, this operation is still secure, as justified by the two reasons stated in the following subsection.

6.3 The avalanche effect

The first reason that CBC-MACs and AES are secure is the *avalanche effect*. The avalanche effect is a characteristic of good encryption algorithm. Basically it states that if two messages are very similar, possibly only different by a single bit, the encryption algorithm's output for both messages will be very different. Algorithms that differ a lot have a strong avalanche effect, which is desirable. Algorithms that differ by a small amount have a weaker avalanche effect. As the algorithm is run over the two similar messages over and over again, the differences in the encrypted output become more and more different. That is how this characteristic comes to be known as the avalanche effect. This term pulls from the idea of a snowball rolling down a hill that gets bigger and bigger until it becomes an avalanche.

The other reason that the CBC-MAC and AES are secure is because the CBC-MAC cannot be faked by an attacker, unless that an attacker is already in the network. We will not consider scenarios in which the attacker has already gained access to the network. This belongs to the intrusion detection issue of the network and should be addressed in a separate paper. In this example, let us assume that the attacker does not know what the keys are to gain access to the WPAN and the attacker is able to eavesdrop on the transmissions and even attempt to disrupt them. These are normal assumptions of the attack model.

This brings us back to the reason why the integrity code cannot be faked. Looking back at Figure 1 and the cipher block chaining mode, one might notice that the last output block of a cipher block chaining encryption is the result of every previous block of plaintext data input to the algorithm. As a result, if an attacker is somehow able to change one bit somewhere in the middle of the message, the avalanche effect of AES will ensure that the change in that one bit will produce a large number of changes in the subsequent output blocks. Consequently, it is statistically impossible and computationally infeasible to attempt to change a part of a message and the attached integrity code and still have them consistent. The only critical aspect to this is that the attacker does not know the key. Without the key to the AES algorithm, changing the data without being detected is an intractable problem for computer science.

6.4 *How beacon messages are secured*

The integrity code is generated based on the key, the nonce, some optional authentication data and some optional message data. While at first glance, it might seem that a secure beacon's information elements and synchronisation parameters would be the message, the truth is that this is passed to the CBC-MAC generation function as part of the authentication data. The authentication data for the Secure Beacon's integrity code generation is everything in the secure beacon from the MAC header to the very last byte in the information elements. If this does not end in an even 16-bytes block, then the remaining blocks are padded with zeros, so that the algorithm runs smoothly.

When referring back to Figure 1, the CBC Mode encrypts the first block of data after XORing the first block of plaintext with an IV. IEEE 802.15.3 does not use an IV or the first block plaintext in the first block's encryption. Instead, IEEE 802.15.3 seeds this encryption process with a nonce in the first block. The nonce is used in this first block to ensure uniqueness. Table 1 illustrates how this first block is created. This block is encrypted with a key and then XORed with the first block of plaintext to start encrypting the actual data. This new, XORed first block of plaintext is encrypted with the key and then XORed with the second block of data. This pattern continues until every block has been encrypted. The first 8 bytes of the last block from this stream of encrypted blocks are used for the integrity code, so that the PNC must append these 8 bytes to the secure frame payload (the information elements and synchronisation parameters). Once the integrity code has been generated, the integrity code is then using AES in counter mode using the group data key.

When a device receives this secure beacon, there are three possible scenarios that could occur. The first possibility is that the receiving device is not a member of the piconet. In this situation, the device may attempt to join the piconet. In the second possibility, the device is a secure member of the piconet, but it does not have a current group data key. When this happens, the device checks the SECID field to verify the key that should be used. If the device does not recognise this particular SECID, then the device knows that it somehow misses a change in the group data key and it will issue a request to the PNC for the current group data key. The final possibility is that the device is a member of the piconet and it has the correct key. In this case, the device, upon receipt of the secure beacon, will generate an integrity code using CBC-MAC on the secure beacon and compare the received integrity code with the newly generated integrity code. If the two values match, the secure beacon has been authenticated.

6.5 *Sending and receiving a data message*

At some point in the lifetime of this piconet, at least one device is going to send and receive a data message. The difference between a secure data message and a secure beacon message is that the secure data message is encrypted and the secure beacon message is not. They both still get an integrity code generated.

When a high-layer application is going to transmit some data to a specific device within the piconet, the message must begin with the MAC header, which includes the ID of the sending device and the ID of the intended recipient. This also has the SEC bit in the Frame Control field set to one to indicate that this message is a secure message and will need to be decrypted and validated. If this message is sent without the SEC bit set, then the recipient could reject the message. Any message sent without security during a secure superframe can be rejected just because the superframe is supposed to be secure.

The next step is generating the integrity code. This is done in a similar way as the integrity code for a secure beacon message. The difference is that data messages actually have a message text, unlike the secure beacon, which uses a zero length string for the message text. For data messages, the additional authentication data is made up of the MAC header, the SECID and the secure frame counter. Since this is a total of 14 bytes, the additional 2 bytes in the 128-bit block are padded with zeros. The message text is composed of the message itself. If the message cannot be divided evenly into 128 bit blocks, then the message is padded with zeros. Once again, the first block, B_0, is made up of a nonce, the length of the message, $l(m)$ and some authentication flags. Table 1 displays the structure of the authentication flags. This first block is critical, because this is the block that identifies how long the message text is and whether or not there are any more additional authentication blocks.

If an upper level application wishes to provide additional authentication data, then that is possible.

The Adata flag in the authentication flag byte is set to 1 in this situation. The next bytes are additional authentication data.

Once the length of the message is known and the length of the authentication data is known, the integrity code is just generated using the same principle as before. The initial block is encrypted using the key. There is a decision to be made about which key to use. Normally, in a device to device transmission, there should be a peer data key shared between the two devices. If that's not available, then the group data key for that piconet can be used instead. If this device is not in the piconet, then peer data keys must be used or the transmission cannot be made secure. Consequently, the device will not send the transmission if the message requires a secure method of communication. Once the initial block is built and the SECID is chosen, the sending device can easily generate the integrity code using the steps from the previous sections.

The function that generates the integrity code is going to take the initial block, the authentication data and the message text. The initial block will contain the length of the message text, the nonce and the authentication flags. Within the authentication flags, the L field will be set to two as specified by the specification's requirements and the M field will be set to two as well, since the 8 bytes integrity code has already been required by the specification as well. Also the Adata bit will be set to one and the reserved bit will be set to zero. In binary, the authentication byte would be 01010010.

Using this data, the integrity code generation function would know that there is authentication data to be included in this integrity code generation and therefore these blocks must be generated and prepared for the function. To generate the authentication data, the size of the authentication data must be known. The size is coded in 2, 4 or 10 bytes, depending on the size itself. The first two blocks determine the size of this data.

For a data message, the required authentication blocks are the MAC header (10 bytes), the SECID (2 bytes) and the secure frame counter (2 bytes) for a total of 14 bytes. Since $l(a)$ equals 14 in this case, $l(a)$ will only use 2 bytes, because the specification stipulates that if $0 \quad l(a) \quad 2^{16} \quad 2^{8}$, then $l(a)$ will only use 2 bytes. This means that the authentication data will use only one 16-bytes block. The first 2 bytes will hold $l(a)$, the next ten will hold the MAC header, the next two will hold the SECID and the final two will hold the secure frame counter. Since this block is an even 16 bytes, there is no need to pad the authentication data with zeros.

Since the message text can be a variable number of bytes, let us assume that the message text is 301 bytes long. This unusual length is specifically chosen for two reasons. Firstly, the length does not divide evenly into 16-bytes blocks, so this message will, if needed, require padded zero bytes. Secondly, the length is greater than 255, so that it illustrates one of the problems in the specification. This discrepancy will be described in the next section. For the purposes of continuing this example, the length requirement will be treated as if the specification correctly represents the length of the message.

Since the message data is 301 bytes, some additional bytes need to be added with zero values before the message data is passed to the integrity code generation function. If 3 bytes, each with a value of zero, are added to the end of the message data, the message data will be 304 bytes long, which equals 19 complete 16-bytes blocks. Finally, the integrity code generation function can generate the 8-bytes integrity code based on the initial block, the one block of authentication data and the 19 blocks of message data. The first 8 bytes of the final ciphertext block becomes the integrity code.

6.6 *Encrypting the message*

Now since the integrity code has been generated, the next step is to encrypt the message itself before it can be transmitted over the wireless channel. Since IEEE 802.15.3 is encrypted using AES in counter mode, the method of generating the counter becomes critical. Table 8 illustrates the format of the counter. This figure is almost identical to Table 1, except that it uses a counter in place of $l(m)$ and it uses encryption flags instead of authentication flags. In fact, the format of the encryption flags is exactly the same. The only difference is that only bytes 0, 1 and 2 can be non-zero, so they hold the value L. In this case, the encryption flags always equal 00000010.

Table 8 Encryption block zero – this figure shows the structure of A_0, which is very similar to Table 1 There is one slight problem with this diagram

2 bytes	*13 bytes*	*1 byte*
Encryption flags	Nonce	Counter

Since the specification uses counter mode, a key stream is generated using this encryption block and incrementing the counter field for each block of data. As a result, the key identified in the SECID field is used to encrypt each counter.

As stated before, the message in this example is 19 blocks long, so the encryption algorithm, CCM, will need to generate 20 blocks of the key stream. This key stream generation produces blocks A_0 to A_{19}. The first block, A_0, is used for encrypting the CBC-MAC. The rest of the blocks are XORed with the message data to produce the ciphertext. Since an attacker can easily disrupt the integrity of a message encrypted in counter mode (i.e. changing one bit of ciphertext switches the corresponding bit of plaintext), an integrity code is also generated to authenticate the message. The integrity code generated earlier is XORed with the first M bytes of A_0. In this way, both the message and the integrity code are encrypted for transmission.

The only question remaining is whether or not the recipient of this data message has all the necessary information to decrypt the message. Obviously, the recipient needs to know the key used, and this should have been exchanged before this message is sent. Therefore, the receiving device can find the key associated with the given SECID. Then, the device needs to be able to form the nonce, the initial encryption block and the initial authentication block, all of which are possible given unencrypted data in the MAC header, SECID and the secure frame counter. Finally, all of the authentication data must be present in order to authenticate the integrity code. If any part of the message has been tampered with, there is a very slim chance that this will not be detected.

6.7 Message encryption specifications

There are some general guidelines on how to secure different messages. This section will just go over the different rules.

For a secure beacon, the integrity code is generated using the group data key with the entire beacon as the authentication data and a zero length string as the message data. Then, the integrity code is encrypted.

A secure command integrity code is generated with either the peer data key or the most relevant and available key. At worst, this would use the group data key. The integrity code is generated using the entire command as the authentication data and an empty string for the message data. This means that the integrity code is generated and then encrypted.

To generate the integrity code and encrypt a data message, basically, the integrity code for a data message is generated by using the MAC header, the SECID and the secure frame counter as the authentication data. The message data is the actual data itself for the integrity code generation. Encrypting the data message simply requires encrypting the data and the integrity code using AES encryption in counter mode.

The last kind of message is the encryption of a key for sending across the airwaves. The authentication data for this message is the same as before. All the bytes up to the actual data make up the authentication data for the integrity code generation. The key itself becomes the message text for the CBC-MAC generation. After the integrity code has been generated, the key and the integrity code are encrypted just like a data message. The only difference between a key encryption and a data encryption is that the key is supposed to be preencrypted, so there is an extra layer of encryption for keys.

7 Security vulnerabilities, errors and corrections, assumptions and other discussions

7.1 Errors and corrections in IEEE 802.15.3

This subsection discusses the apparent discrepancy between Table 1, which is consistent with the specification's corresponding figure and the specification's stipulations about the length of the message text and how many bytes should be used to represent this number. Furthermore, Table 8 has the same problem as Table 1, since the counter should be able to be unique for each block of data within the message and will also need to be able to represent the length of the message data.

In Table 4, the L bits are supposed to represent the number of bytes required to represent $l(m)$. The value of L in the specification is two since the restriction on L stems from the length of the nonce. The nonce is 13 bytes, shown in Table 1. The specification stipulates that the length of the nonce, in Section 10.4.1 table 66 (IEEE, 2003), is $15 \quad L$ bytes. Through simple algebra, $L \quad 2$. The specification also stipulates in Section 10.4.1 (IEEE, 2003) that $l(m)$, the length of the message in bytes, has a length restriction of:

$$0 \leq l(m) < 2^{8L} \tag{5}$$

In other words, the maximum value for $l(m)$ must be less than 2^{16}, which requires at least 2 bytes to be able to represent. The rule for representing a number in binary is that N bits can be used to represent a number $2^N \quad 1$. Since the limit for $l(m)$ is $2^{16} \quad 1$ (or 65535), 16 bits are required to represent the full range of $l(m)$. Tables 1 and 8 show the length of the message and the counter, respectively, which should also be big enough to represent the length of the message. However, in Tables 1 and 8, they are both only 1 byte long.

In our opinion, the field for the length of m and the field for the counter should be 2 bytes long and the fields of the authentication or encryption flags should be 1 byte each. There is another reason to set these flag fields only 1 byte long instead of two: nothing is gained by having 2 bytes repeating the same thing. The authentication flags field indicates how many bytes are used to represent, M and L and it indicates whether authentication data is included or not. There is no reason to repeat these values a second time, since that would be redundant. This holds true for the encryption flags field as well.

Our proposed corrections for Tables 1 and 8 are listed in Tables 9 and 10, respectively for the formats of the new zero blocks. Table 9 shows that the authentication flags and the length of m have switched places within the block. Table 10 shows the same thing with the counter and the encryption flags.

Table 9 Possible corrected authentication block zero – this figure shows the structure of what B_0 might truly be in order to represent the length of the message correctly

2 bytes	*13 bytes*	*1 byte*
Length of *m*	Nonce	Authentication flags

Table 10 Possible corrected encryption block zero – this figure shows the structure of what A_0 might truly be in order to represent the length of the message correctly

2 bytes	*13 bytes*	*1 byte*
Counter	Nonce	Encryption flags

7.2 Security assumptions

There are a number of assumptions made for the IEEE 802.15.3 specification. The specification committee assumes that transmissions in the piconet can be eavesdropped by devices other than those in the piconet. The devices using IEEE 802.15.3 may not have much power to use for cryptographic computations, so that the algorithms are chosen to be efficient for small devices. The specification has the assumption that there is no access to an external network, so that authentication servers are not mentioned. Another assumption is that a device could leave the network at any time, so that the specification allows for devices to be suddenly lost.

The specification also assumes that attackers will not only have the ability to listen to a piconet's transmissions, but also be capable of performing state-of-the-art cryptographic attacks. In addition, the attackers are assumed to be able to make synchronised transmissions.

7.3 *Security vulnerabilities*

There are several security vulnerabilities that are not solved by this specification.

7.3.1 *Denial of service and selective denial of service*

First, anyone who can transmit radio signals can cause a denial of service attack in a piconet. By sending ones out into the wireless medium at the proper frequencies, a secure piconet is guaranteed to reject every message sent, because every message would fail its authentication step. Essentially, communications between devices is shut down. Unfortunately, this kind of attack is almost immature, because it is obviously noticeable and nothing can be gained through such efforts except a temporary loss of service to the wireless devices until the source of the noise is located and eliminated. This vulnerability, however, is a weakness of every wireless network, because there is currently no way to prevent radio waves from being disrupted in such a manner. The laws of physics prevent this. However, smart attackers can easily launch selective denial of service attack, that is, attackers selectively send out frames in some occasions so that denial of service or another malicious goal can be achieved and in the mean time, they avoid being caught or noticed.

7.3.2 *Lacking how to join a secure piconet and how to generate keys*

Another major vulnerability is that the IEEE 802.15.3 specification does not define a suitable method for joining group membership, as well as how to generate the appropriate keys, which is left up for vendors to establish. With a step-by-step instruction on how to join a piconet and generate the appropriate keys, a developer could implement an immediate IEEE 802.15.3 ready device and software. The first meaningful event is likely to be a secure beacon message transmitted by the piconet coordinator. It is best to start with this message, because the secure beacon will start the superframe, which governs the time management of the piconet. The lack of a specification for how a device can join a piconet and establish keys makes the specification somewhat dangerous. The specification requires much unique functionality, but there is little or no direction on how some of these requirements might be accomplished. For example, the specification states that devices might be able to join a piconet and establish keys once associated, but this critical process is purposefully left up to the vendors to implement. While this may be the result of a good lesson learned from IEEE 802.11, it opens the door for poor vendor implementations to leave something this important unspecified. Furthermore, the task of figuring out how to establish a secure relationship between two devices that have never met is also left up to the vendors. The most difficult parts of IEEE 802.15.3 are the problems that are not solved by the specification writers.

7.3.3 *Key life cycles*

The life cycle of a key is an important security consideration, because a key becomes less secure after it is used for a long time. This is due to the fact that attackers may have more opportunities. The specification does not specify many restrictions on the life cycle of any particular key, but it does provide the mechanisms for enforcing restrictions imposed by a device.

So far, the only key life cycle restriction specified is that the group data key must be changed anytime there is a change in the group membership. One requirement not previously mentioned is that IEEE 802.15.3 requires that a device must have the ability to require a secure frame to be transmitted by any device with which it has a secure relationship, but it does not specify how this is enforced. A device can require that a device send a secure frame within a certain amount of time. If the device does not send the frame, then the requesting device can disassociate or terminate the relationship.

7.3.4 *Freshness protection and attacks*

Each device must do two things to ensure freshness protection. First, the device must keep track of the previous time token sent in the secure beacon at the start of each superframe. The device must also be sure to check the group data key SECID. If the SECID does not match the device's current group data key, then the device will issue a Key Request message to the PNC. With the current group data key, the device can verify the integrity code sent with the secure beacon and verify that the secure beacon's data is authentic. If the time token received is not greater than the previous time token, then the message is rejected.

However, attackers can easily launch denial of service attacks, by sending incorrect SECID or incorrect integrity code to make the device keep sending Key Request messages, etc.

7.3.5 *Same nonce attacks and same key attacks*

Firstly, since the nonce is only 13 bytes in length, that is, 104 bits, it is possible that nonces are repeated. Furthermore, the same nonce could happen when the devices are restarted due to some reasons such as battery failure since nonces are more likely starting some small values such as zeros, but this can be prevented if the initial nonce is generated by a good random number generator. However, if the random number sequence uses the same initial vector or seed, pseudorandom numbers are not random when the devices are restarted.

Secondly, since key-generations are not specified in the IEEE 802.15.3 specification, different vendors may adopt different key-generation approaches for different devices. Therefore, it is possible that a key is used again.

The IEEE 802.15.3 specification stipulates that there can be no repetition of a key – nonce combination. In other words, if a key is to be used again, then there must be a different nonce and vice versa. However, due to the above two discussions, it is possible for attackers to launch the same nonce and same key attacks.

7.4 Other discussions

7.4.1 Discussions on avalanche effect

It is important to note that the counter mode encryption is not very strong by itself, because any change in a bit of the ciphertext results in a change in only a bit in the plaintext.

The main strength of the CBC-MAC scheme is that every block is dependent on the previous block's encryption so that it is very difficult to tamper with the data without the manipulation being detected. This method makes the authentication very secure, because even a small change in one bit in the data will cause a large change in the final block due to the avalanche effect. The main weakness of this method is that it *cannot be done in parallel*. This method requires that a block be fully encrypted before the next block can be encrypted.

Because AES has what is referred to as the *avalanche effect*, even a change of a single bit will cause the integrity check to fail when the message is decrypted.

7.4.2 Discussions on counter

One might think that the counter could only be used for up to 255 blocks, because the counter only uses 1 byte; however, the specification does not mention any restriction to this effect directly.

7.4.3 Discussions on PNC

The specification does leave room for the existing PNC to send information to the devices in the piconet before the actual transition of the PNC role, but any of these methods would be vendor specific. The specification does not specify any necessary action for this part, but merely opens the door in the protocol.

7.4.4 Discussions on security setup

There are no assumptions made about whether an attacker is present or not during the security setup, which could occur before or during the establishment of a piconet.

8 Conclusion

The unique characteristics of wireless networks make the provision of security mechanisms a non-trivial task. This paper has provided a general introduction to the security issues of the IEEE 802.15.3 specification. It is a survey on the various security aspects and mechanisms in the IEEE 802.15.3 WPANs. It provides security analysis, and points out some errors and many security vulnerabilities of these networks. We hope that this paper will trigger more research development to the open vulnerabilities as described in this paper.

References

Chen, X., Xiao, Y., Cai, Y., Lu, J. and Zhou, Z. (2006) 'An energy diffserv and application-aware MAC layer scheduling for multiple VBR video streaming over high-rate WPANs', *Computer Communications, Elsevier Science*, Vol. 29, No. 17, pp.3516–3526.

IEEE (2003) *Specification for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements*, IEEE Specification 802.15.3.

Stallings, W. (2003) *Cryptography and Network Security: Principles and Practices*, 3rd edition, Upper Saddle River, NJ: Pearson Education, Inc.

Xiao, Y. (2005) 'MAC layer issues and throughput analysis for the IEEE 802.15.3a UWB', *Dynamics of Continuous, Discrete and Impulsive Systems--An International Journal for Theory and Applications (Series B) -Special Issue: Ultra-Wideband (UWB) Wireless Communications*, Vol. 12, No. 3, pp.443–462.

Xiao, Y., Shen, X. and Jiang, H. (2006) 'Optimal ACK schemes of the IEEE 802.15.3 MAC for the ultra-wideband system', *IEEE Journal on Selected Areas in Communications*, Vol. 24, No. 4, pp.836–842.