AVR2027: AES Security Module

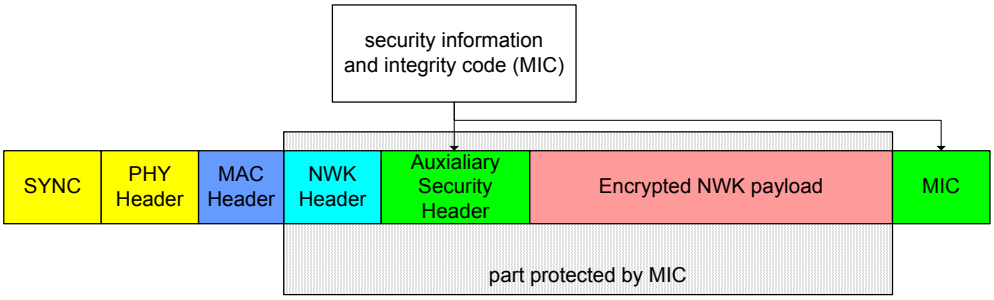
Features

- **Overview of IEEE 802.15.4™, cryptography**
  - **Security of algorithm and protocol**
  - **Encryption modes**
  - **Implementation issues**
- **Using the AES security module in IEEE 802.15.4 and other contexts**
  - **Register description**
  - **Random number generator usage**

1 Introduction

This document briefly reviews IEEE 802.15.4 /ZigBee® cryptography and shows how the AT86RF231's and AT86RF212's security modules simplify the implementation of these features.

Figure 1-1. Format of an encrypted network layer frame



8-bit AVR® Microcontrollers

Application Note

Rev. 8260A-AVR-09/09

2 Overview of IEEE 802.15.4 cryptography

2.1 Scope of IEEE 802.15.4 cryptography

2.1.1 Encryption

Encryption is used to protect information to be read by some unauthorized third party, especially if a message is send over an unsecure channel. There should be no danger if a third party knows the implemented algorithm. The security must be based on a secret key only. The IEEE 802.15.4 standard uses AES-128 (Advanced Encryption Standard) with a 128 bit key length encryption. To the best knowledge of cryptographic research, this algorithm satisfies all modern security requirements; see [1], [2], [3].

However, the wrong application of a good algorithm may destroy security nevertheless. This can be avoided by so-called encryption modes, i.e. how a cryptographic algorithm is applied. Details are explained in 2.2.2.

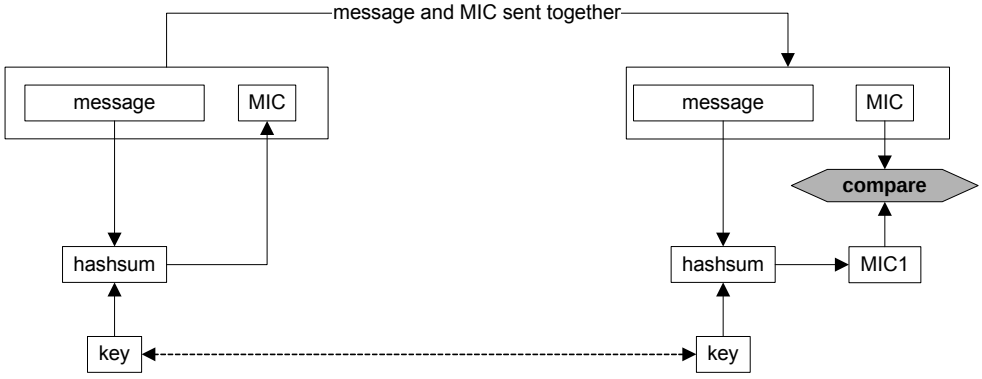
2.1.2 Integrity protection

Encryption is a defense against passive attacks, i.e. against eavesdroppers. But it is often more important to protect yourself against active attackers who send faked or maliciously modified messages. So it is usually no secret whether a window is open or closed, i.e. such a message needs not to be encrypted. However, it is very important that no attacker can send a notification "window is closed" to an alarm system when the window is just open. All this especially applies to wireless sensor networks, since radio messages are easy to fake.

Cryptography can help to detect unauthorized sent or modified messages by appending cryptographic checksums, so-called MACs (Message Authentication Code), in this context named MICs (Message Integrity Check). They can only be computed with the knowledge of a secret key as is shown in Figure 2-1: The sender computes the MIC and sends it together with the message. The receiver re-computes the MIC from the received message and compares it with the MIC in the message. If computed and received MIC coincide, the sender must have known the secret key. An attacker not knowing this key would not be able to modify the message and to compute a valid MIC. Thus the MIC guarantees that the message was generated by the sender and not by some attacker, provided that the secret key was not leaked.

There is an important difference between MICs and CRC checksums which are widely used in hardware: A CRC protects against unintended modifications during transmission, but not against active attackers. CRC checksums do not use keys and are easier to compute than MICs.

Figure 2-1. A MIC detects modifications



The IEEE 802.15.4 selects the so-called CCM* encryption mode which also allows to enable integrity protection.

Integrity protection has some important consequences:

- The message payload can not be modified by some attacker without getting noticed.
- If the sender ID is included in the MIC computation, the receiver can be sure who sent this message, i.e. spoofing is excluded.
- When an increasing frame counter is included in the MIC computation, replay attacks are excluded. That is, a message recorded by some listener would not be accepted again (e.g., "window is closed").
- By the same mean, the right order of sent messages can be ensured (think of "window open" - "window closed"). It is possible that the message order is changed by routing problems.

For critical applications, also "delay attacks" can be avoided when time stamps are included in the MIC computation. In such a scenario, the attacker catches a message and disturbs the receiver at the same time. The message is then sent later.

2.2 Cryptography used in IEEE 802.15.4

2.2.1 The AES algorithm

Encryption according to the IEEE 802.15.4 standard is based on the AES algorithm. Though security of usable algorithms has not be proven yet, AES was selected as new encryption standard in a world-wide competition and has been investigated by the best cryptanalysts for years, who did not find any weakness in it. So AES can be considered as one of the best available cryptographic algorithms. It replaces the 30 year old DES which uses a 56 bit key only. Whereas all possible 56 bit long keys can be tried by special hardware now, the 128 bit key of AES will never guessed with hardware as we know it today (and even the hypothetical quantum computers could not break at least 256 bit keys using recent theory) – see [5].

AES is hardware-friendly: It requires relatively few resources, and it is fast. So it is the best choice for symmetric cryptography in low power devices.

AES can use 128, 192 and 256 bit long keys. Even 128 bit keys cannot be broken; longer key lengths are a precaution if still unknown weaknesses of the algorithm would be found. IEEE 802.15.4 uses AES-128, i.e. 128 bit keys.

AES is a block cipher: Encryption and decryption are done on portions of the same length only, here 128 bit (16 byte), yielding a 128 bit result - for all key sizes. How to deal with lengths of plain or ciphertext which are no multiples of 16, is a matter of encryption modes (see section 2.2.2).

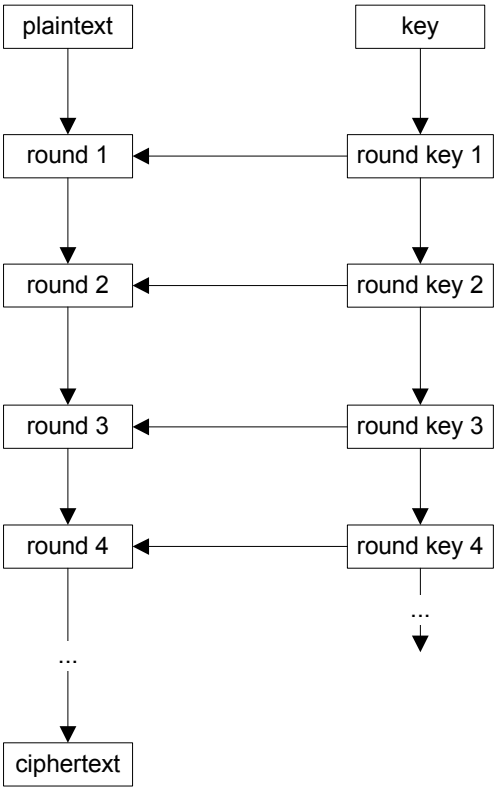
Moreover, AES is a so-called product cipher (see Figure 2-2):

- The first step is the key initialization: Based on the key, 10 round keys are computed.
- After this, encryption/decryption is done in 10 almost identical steps (rounds), each depending on the corresponding round key. Each round consists of rather simple transformations like bytewise XOR, byte substitution and byte permutation.

In contrast to some other algorithms, the decryption procedure differs from encryption. For IEEE 802.15.4, this does not matter since in CCM* mode, even CCM* decryption only uses AES encryption (see section 2.2.2.6). Nevertheless, the radio transceiver

security module also offers AES decryption functionality. To this end, the last round key must be computed before and then used as key (see section 3.6).

Figure 2-2. Structure of a product algorithm



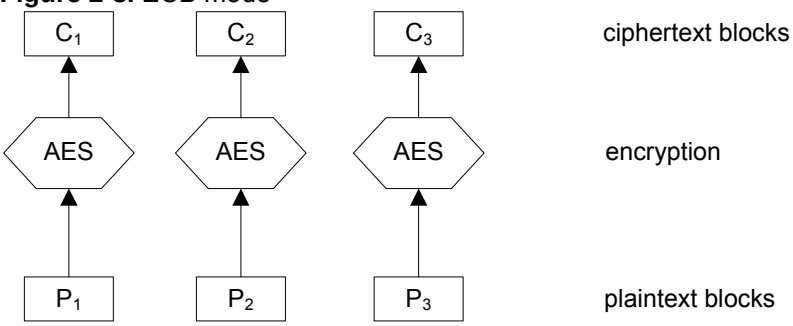
2.2.2 Encryption modes, CBC-MAC, padding

A cryptographic algorithm can encrypt or decrypt one block only. Encryption modes deal with how to apply a block algorithm to messages of arbitrary lengths [6]. This is not only a technical question, but it has also great influence on security. Only topics of importance to IEEE 802.15.4 are discussed in the following.

2.2.2.1 ECB mode

The simplest way is the ECB (electronic code book) mode: Each plain or ciphertext block (in our case, 16 byte) is en- respectively decrypted separately with the same key, as shown in Figure 2-3.

Figure 2-3. ECB mode

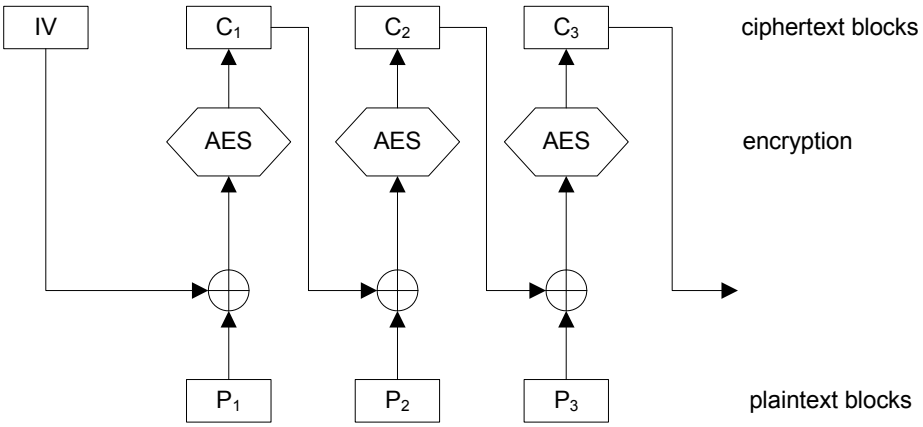


This mode is simple to implement and allows parallelization. However, it does not fully conceal the plaintext structure (e.g., a plaintext consisting of null bytes only would encrypt to a sequence of identical 16 byte blocks). In particular, identical plaintexts yield the same ciphertexts what can allow practical attacks: Suppose a sensor has two states "on" and "off", i.e. generates two kinds of messages only. If the messages are encrypted straightforward, i.e. in ECB mode, an attacker also observes two kinds of encrypted messages (ciphertexts) only. It is usually not hard to guess from the context which ciphertext belongs to which message. Therefore ECB is applied in special cases only, e.g. to encrypt other (random) keys. In general, a more secure mode should be applied, like the CBC mode explained in the following paragraph.

2.2.2.2 CBC mode

The most important mode in practice is the CBC (cipher block chaining) mode, as shown in Figure 2-4: During encryption, the last computed ciphertext block is XOR'ed with the actual plaintext block and then encrypted. The first block, the so-called initialization vector (IV), is pseudo-random and XORed with the first plaintext block before encryption.

Figure 2-4. CBC mode



CBC completely hides information about plaintext (with exception of the length, of course), even the fact two ciphertexts belong to the same plaintext - provided that different initialization vectors are used for different encrypted messages.

2.2.2.3 CBC-MAC

The CBC mode can be used for MAC computation: Since the last computed ciphertext depends on all plaintext blocks, it is suitable as cryptographic checksum (MAC) which can be computed only if the secret key is known. Usually, the initialization vector is 0 in this case. For fixed length messages, the CBC-MAC is secure.

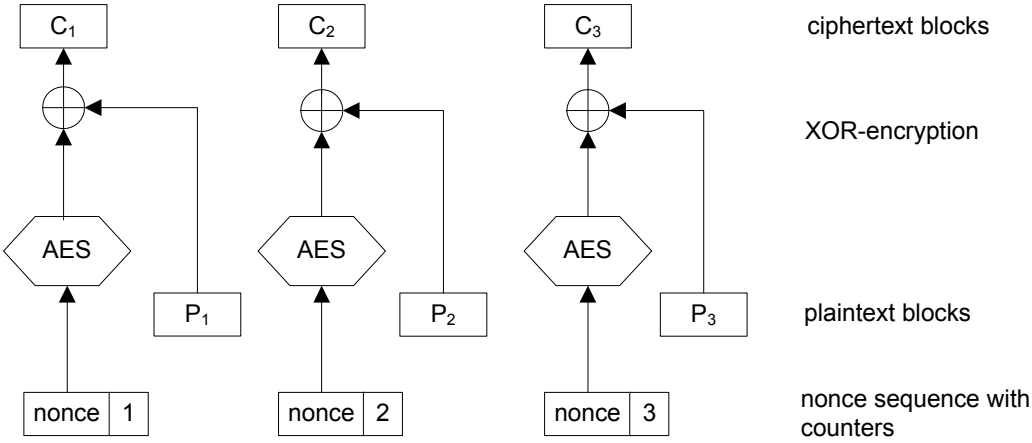
IEEE 802.15.4 uses the CCM* mode, and this computes CBC-MACs of variable length messages. However, the first message block is a so-called nonce containing the message length what implies that the computed CBC-MAC (called MIC in CCM*) is also secure.

2.2.2.4 The counter mode

The counter mode (CTR) works as shown in Figure 2-5: A sequence of nonces (unique blocks of known contents) is usually constructed by reserving some portion of a fixed nonce and filling this part with an increasing sequence of numbers. This stream of

nonces is encrypted block by block, and the resulting ciphertext stream is bitwise XORed with the plaintext giving the ciphertext.

Figure 2-5. Counter mode (CTR)



In spite of its simple construction, the CTR mode is secure if integrity protection applies, and if no combination of nonce with counter is repeated for the same key. The mode offers some useful features:

- It can be parallelized.
- Decryption is exactly the same operation as encryption.
- No padding is required (see below).
- Ciphertexts depend on nonces, i.e. identical messages can be encrypted to different ciphertexts.

The counter mode acts like any stream cipher where a bit stream is simply XOR'ed with the plain respectively ciphertext. For such stream ciphers, there is some important pitfall:

- Messages can be modified by an active attacker, even if he cannot decrypt them. This must be prevented by included integrity protection, what is sometimes overseen.

2.2.2.5 Padding

All modes explained up to here apply to messages with multiples of block length (for AES-128: 16 byte) only. There are different methods to circumvent this problem (see [4], 9.1 and 9.3, and [5], 5.1.2). For the purpose of this document, the simplest method will suffice: Padding, i.e. appending as many (possibly none) bytes to the message until a multiple of the block length is reached. Usually, null bytes are taken. In general, decryption will then not be unique if the last byte of the plaintext can be a null byte (since the end of the plaintext cannot be determined in a unique way). The CCM* mode described in IEEE 802.15.4 contains the message length in the nonce (which is contained in the first authenticated message block) and thus solves this problem.

2.2.2.6 The CCM and CCM modes*

The CCM mode is a combination of CBC-MAC and CTR mode to combine a cleartext header with an encrypted payload where header and payload together should be integrity protected. It was published as NIST standard and as RFC3610 [7], [8], [9]. The security can be proven for any block cipher supposed that the block cipher is secure.

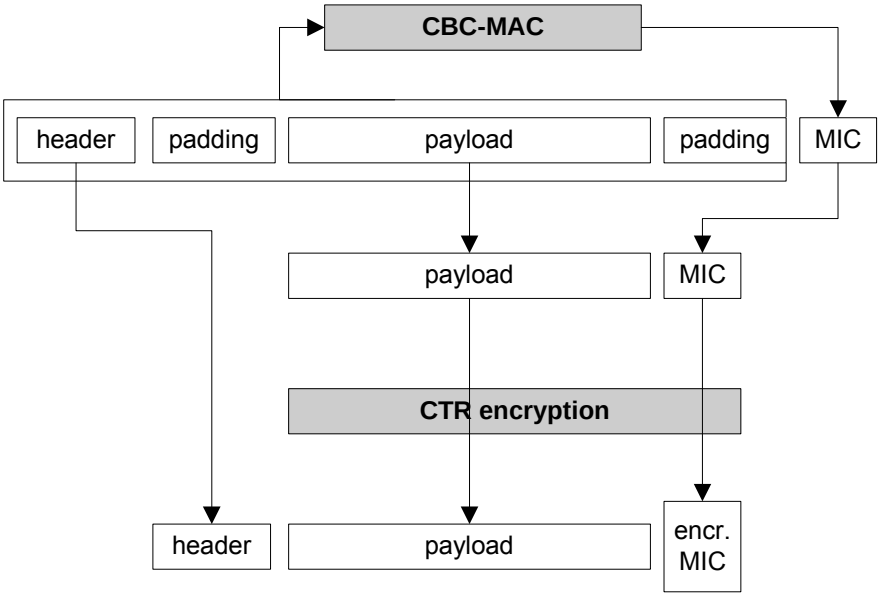
CCM* differs from CCM in that encryption only and authentication only are allowed.

The CBC-MAC used in CCM/CCM* can have reduced length and is called MIC there.

A CCM securing as used in IEEE 802.15.4 is done as follows (see also Figure 2-6):

- A nonce is constructed containing lengths of cleartext header and payload and prepended to the message. A nonce is a part of an AES block which contents must be unique over all messages secured with the same key (this step is not shown in the figure).
- After this, cleartext header (containing the nonce) and payload are padded with null bytes separately and concatenated.
- Then a CBC-MAC (MIC) is computed over the whole message with initialization vector 0 and appended to the concatenation.
- Then the nonce used for the CBC-MAC (MIC) computation is changed.
- Then payload and MIC are encrypted in CTR mode with the new nonce where the counter for the MIC encryption is 0, and the counters for payload encryption start with 1.
- Finally, the MIC can be reduced in length. MIC lengths of 0, 32, 64 or 128 bit are allowed (0 bit length means: no authentication, the MIC is not computed).

Figure 2-6. CCM mode



For technical details, see [8], [9] and certainly the IEEE 802.15.4 standard [11].

2.3 Pros and cons of IEEE 802.15.4 cryptography

Pros:

1. Only symmetric cryptography is used which is much faster, far easier to implement and easier to handle than asymmetric cryptography (public key cryptography).
2. Because of CCM* mode, only AES encryption is used, no AES decryption is necessary. This allows simpler (and smaller) software and reduced encryption hardware. (In the case of Atmel radio transceivers, AES decryption hardware is available, but it requires extra operations what plays no role if only CCM* is used.)
3. The same key is used for authentication as well for encryption, without compromising security (see [18]). Thus key initialization is rare, firmware becomes smaller and faster. Even in the ZigBee commercial mode (which is not in the scope

of this application note - see [12]), no key has to be re-initialized as long as a device communicates with a fixed partner.

4. The cryptography part of the standard is rather simple. This is not only nice for firmware development but also for security: Good security must be compact and easy to review.
5. Algorithm and encryption mode are both free of patents.

Cons:

1. Key transfer is risky: To transfer a secret key over air, there is no other way than to encrypt it by a master key which has to be distributed during initialization by out-of-band means. If this master key is compromised (e.g. since a device was stolen and analyzed), most security might be lost.
2. CCM* allows encryption without authentication. This mode should never be used since it is insecure. It should have been excluded in CCM*. At least, ZigBee does not recommend it.
3. To compute a MIC and encrypt a message, the message has to be encrypted twice: once for the MIC computation, and once for the encryption itself (and the MIC encryption requires one more block to be encrypted).

   There are almost twice faster modes to combine message authentication with payload encryption, but they are not patent-free [16].
4. MIC encryption is not necessary - because of the nonce structure, a listener can neither gain information about possibly identical payloads, nor are so-called collision attacks feasible. This was also criticized in [13].
5. Generally, CCM* is considered to be bad designed, other modes like the EAX mode are said to more elegant (for details, see [10], [13]).

Remarks:

1. Implement the standard:

   The mentioned drawbacks are not as dramatic as it seems. The implementation of CCM* is, once done, quite compact and easy to check with the help of test vectors. The only theoretical obstacle could be item 1. of the "cons" list, i.e. the performance. For the Atmel radio transceivers however, this may not matter, dependent on the application: Since sending one AES block over air requires about 500 µs in the 250 Kbit/s mode, the time of 130 µs needed for two encryptions and two SPI transfers (see section 3.8) is marginal.

   The win of a slightly shorter and/or more elegant code does not outweigh the high price of incompatibility with a standard. So it is recommended to implement the CCM* mode (without the encryption only option) as described in IEEE 802.15.4.

2. Comparison to public key cryptography (PKC):

   The problem of secure key transfer is most securely solved by asymmetric cryptography, i.e. using public and private keys. However, the price for this can be quite high:

   - In software, symmetric cryptography can be faster than asymmetric one e.g. by a factor of 1000. So PKC is suited for rare key exchanges only.
   - To achieve reasonable response times, specialized and expensive hardware is required when PKC implemented in software is too slow. Effective algorithms based on ECC (elliptic curve cryptography) - and not the well-

known RSA or Diffie-Hellman algorithms - are patented if they are fast in hardware. The implementation requires a sound theoretical background.

- A PKI (public key infrastructure) requires a lot of resources (especially RAM).
- Symmetric cryptography has to be implemented anyway since it is the "workhorse" for securing package transfer.

PKC would only be needed for secure key transfer what is a rare event in most networks. However, even the simple CCM* mode using the same fixed key for all devices is not implemented or not active by default in many recent systems. Moreover, there are other more important problems like secure routing, or the fact that e.g. ZigBee requires in the simple residential mode (all devices using the same key) a complete unsecuring and securing of each package during each hop.

3 Using the radio transceiver security module

Much of the chapters 3.1 - 3.6. is also contained in the AT86RF231 [14] and AT86RF212 data sheets [15].

3.1 Overview

3.1.1 Functionality

The radio transceiver contains a standalone AES encryption and decryption unit. The following operations are possible:

- Initialize an AES key which is stored locally in the AES unit.
- Choose between encryption and decryption.
- Choose between hardware support for ECB or CBC mode (see section 2.2.2).
- Transfer a plain or ciphertext block to the AES unit.
- Start the encryption/decryption.
- Test whether the encryption/decryption was finished.
- Obtain an en- respectively decrypted block from the AES unit.
- Read the last generated AES round key from the AES unit (needed for decryption).
- Test whether an error occurred in the AES module.

To support full CBC mode or CCM* mode, additional software is required (see sections 3.4, 3.7).

3.1.2 Prerequisites

Controlling the security block is implemented as a fast SRAM access over SPI to address space 0x82 to 0x94. A Fast SRAM access mode allows simultaneously writing new data and reading data from previously processed data within the same SPI transfer and thus reduces overhead in ECB mode (see section 2.2.2.1). In addition, the security module contains another 128-bit register to store the initial key used for security operations. This initial key is not modified by the security module. All registers and commands are explained in chapter 4.1.

The AES unit can only work if the clock module CLKM is activated (subregister CLKM_CTRL != 0 in register 0x03).

3.2 Initialize the key

Key initialization must be the first operation. To set the key, write the value "KEY" (i.e., 1) to the subregister AES_MODE of register AES_CTRL mode and then the 16 byte key to the AES_KEY space (addresses 0x84 - 0x93).

The operation is finished when all data is written. No time for round key generation is needed. As long as the device does not sleep and is not reset, the key is kept in the AES unit and can be used for consecutive encryptions.

3.3 Encryption in ECB mode

To encrypt one block, the following steps are required:

1. The key has to be initialized (see section 3.2).
2. Choose encryption and select ECB encryption mode. To this end, set the subregister AES_DIR of register AES_CTRL to encryption and subregister AES_MODE of register AES_CTRL to ECB.
3. After this, write the input data (the plaintext) to the AES_STATE address space (addresses 0x84 - 0x93). The AES module is still idle in this moment.
4. Start the AES operation: Set the subregister AES_REQUEST of register AES_CTRL to 1. Round keys are computed within this cycle.
5. Either wait 24 µs until the AES block has finished, or poll the subregister AES_DONE of register AES_STATUS until it is 1.
6. Read the result (the ciphertext) from the AES_STATE address space (addresses 0x84 - 0x93).

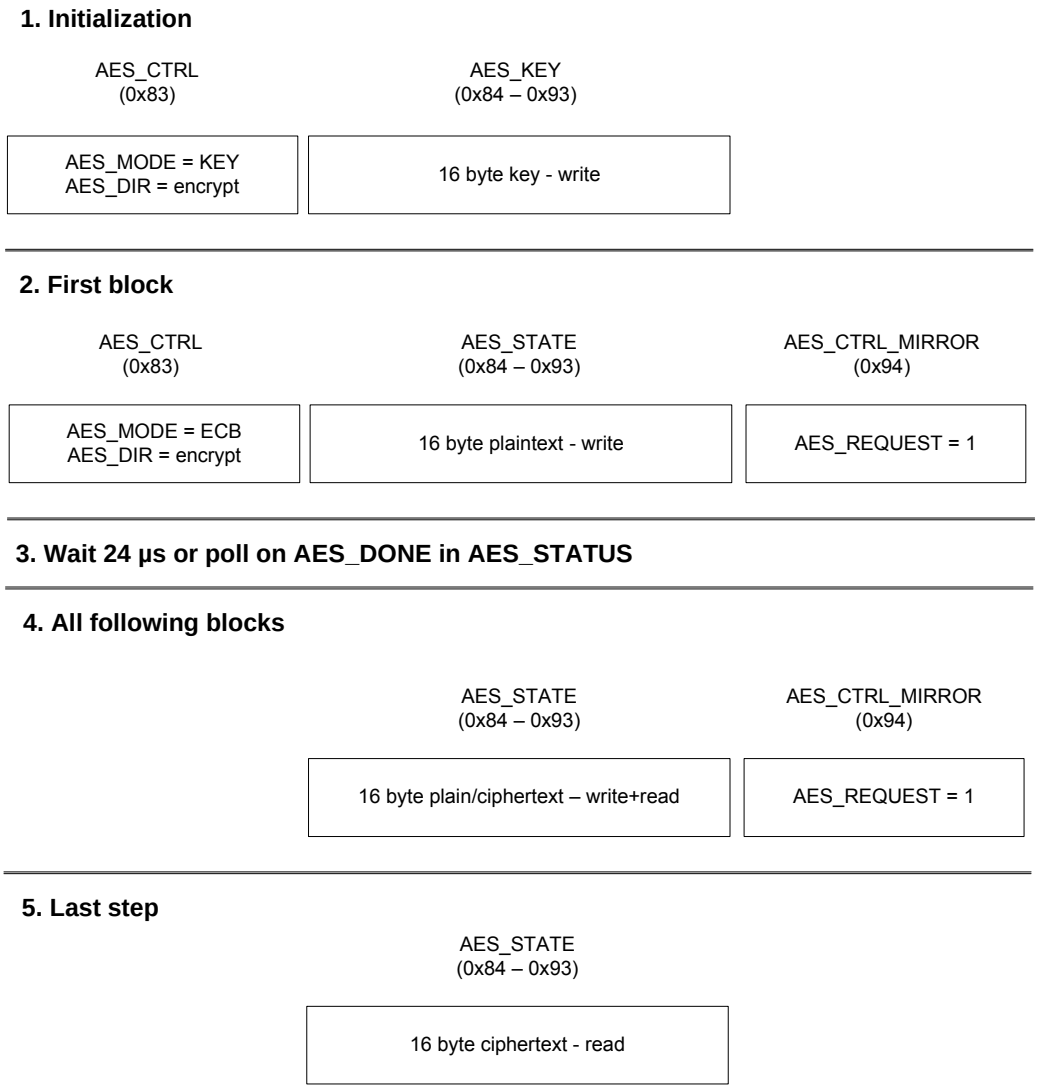
In practice, steps 2 - 6 can be simplified:

The register AES_CTRL_MIRROR has the same function as AES_CTRL. So the data of AES_CTRL in step 2, the AES block, and AES_CTRL_MIRROR (with the AES_REQUEST bit set) can be concatenated to one 18 byte block and written within one SPI access to the AES unit. Moreover, reading and writing of the SPI interface are done simultaneously using the so-called Fast SRAM access for the registers used here (0x82 to 0x94).

Thus the operation reduces to 5 steps (see Figure 3-1):

1. Initialize the key.
2. For the first block, write in one SPI read/write access the following 18 byte, starting with address of AES_CTRL (0x83):
    a. register AES_CTRL as described in step 2 above,
    b. the first block of plaintext (16 byte),
    c. register AES_CTRL_MIRROR with the same contents as register AES_CTRL but additionally subregister AES_REQUEST set to 1.
3. Wait for the result (see step 5 above).
4. From the second block, only write data and AES_CTRL_MIRROR, then goto step 3.
5. After the last block, read the last ciphertext.

Figure 3-1. ECB encryption with the AES unit



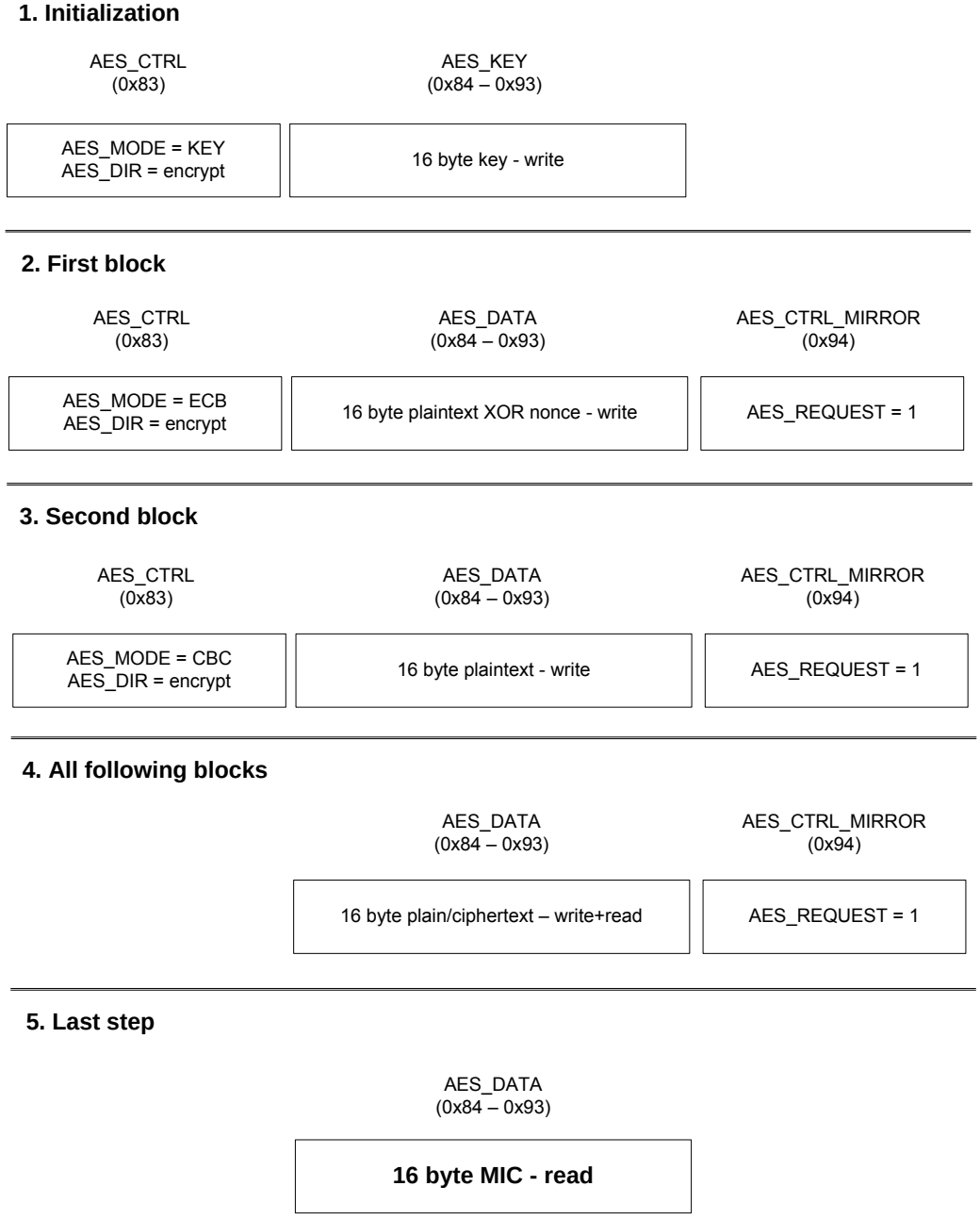
3.4 Encryption in CBC mode

If the AES unit runs in CBC mode, the plaintext is bitwise XORed with the result of the last encryption. So to compute a CBC MIC, the following steps are required:

1. Compute the bitwise XOR of the initialization vector IV (nonce) with the first plaintext.
2. Write it to the AES unit and encrypt it in ECB mode.
3. All following blocks are encrypted in CBC mode of the AES unit.
4. Read the last block from the AES unit. This is the MIC.

In the case of CCM* in IEEE 802.15.4, step 1 is omitted since the initialization vector is 0, so XORing it with the first block changes nothing and is skipped (see Figure 3-2).

Figure 3-2. CBC-MIC computation with the AES unit (see also Figure 3-1)



Remark: For simplicity, padding is not regarded here.

Remark: For CBC decryption, the XOR operation has to be done *after* encryption with the last ciphertext and thus must be done in software using the ECB mode (the CBC operation mode does the XOR *before* encryption).

3.5 Implementing CTR mode

Encryption in CTR mode has partially to be done in software, i.e. the nonce computing and the bitwise XOR with the plaintext. For CCM* encryption in IEEE 802.15.4, the counter part of the nonce can have values of 0 ... 7 only, since a message is maximally

127 byte long. Thus the message consists of at most 8 AES blocks, and increasing the counter means changing one byte in the nonce only. The sequence of nonces has to be encrypted in ECB mode (see section 2.2.2.1).

3.6 Decryption (example for ECB mode)

The AES algorithm requires different keys for encryption and decryption. The decryption key (what is the last generated round key) can be read from the AES_KEY space after any encryption.

Hence to decrypt an ECB encrypted message, the following steps are required:

1. Initialize the AES unit with the original key (see section 3.2).
2. Start a dummy encryption by setting the subregister AES_REQUEST of register AES_CTRL to 1.
3. Read the AES_KEY registers.
4. Initialize the AES unit with this key.
5. Set the encryption mode to ECB (subregister AES_MODE of register AES_CTRL) and AES operation direction to "decryption" (subregister AES_DIR of register AES_CTRL).
6. Write the ciphertext to AES_STATE registers.
7. Start the encryption by setting subregister AES_REQUEST of register AES_CTRL to 1.
8. Wait 24 µs or poll the subregister AES_DONE of register AES_STATUS until it is set.
9. Read the result from the AES_STATE registers.
10. If necessary, go to step 6.

As in the case of ECB encryption, several steps can be joined in one SPI write operation (see section 3.3).

The decryption in CTR mode is identical to the encryption and thus does not require any precaution.

3.7 Implementing CCM*

Hardware support for CCM* encryption is limited to the CBC and CTR encryption as described in 3.4 and 3.5. Assembling the nonce has to be done in software anyway, as well as padding and extending the payload by security header and appending the MIC.

Details can be found in [11], Appendix B.4.

3.8 Timing

3.8.1 CCM mode timing estimation

Using an SPI clock of 4 MHz, the Fast SRAM access to encrypt one AES block lasts about 40 µs (18 byte for data and control - see section 3.3 -, and 2 byte for SPI control). The encryption itself lasts 24 µs. The bytewise XOR of two AES blocks for an 8 MHz AVR clock lasts about 30 µs.

A full CCM* encryption of a message of n AES blocks length and with maximal MIC size (i.e., 16 byte) takes at least the following time:

MIC computation:

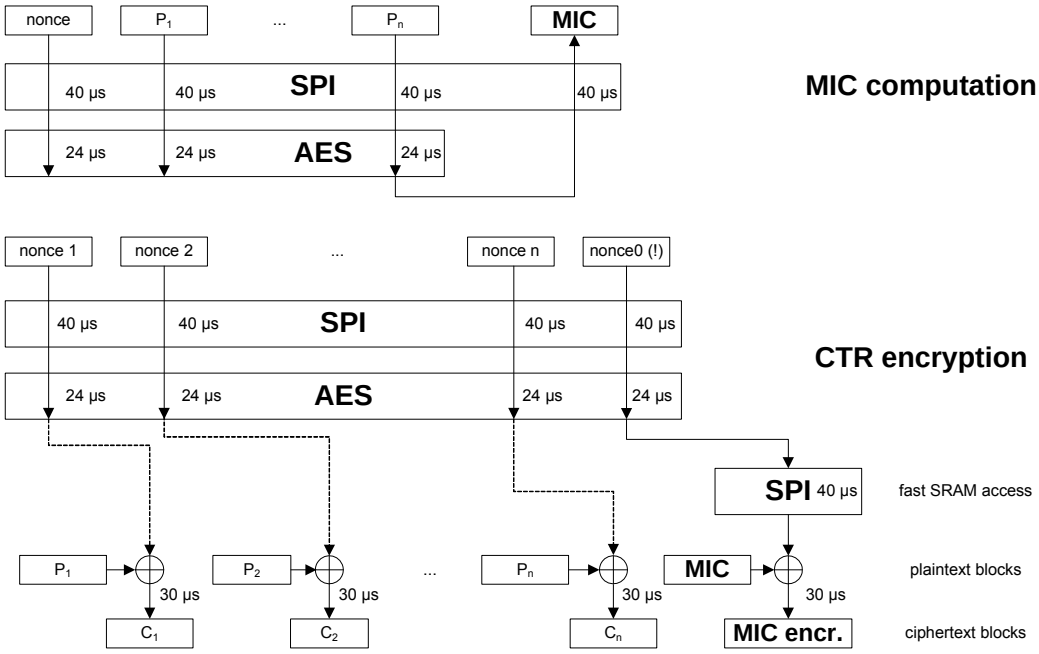
- (n+1)*40 µs for the SPI transfer of nonce and message blocks
- (n+1)*24 µs for encrypting nonce and message blocks
- additional 40 µs for reading the MIC from the AES unit

CTR encryption:

- (n+1)*40 µs for the SPI transfer of counter blocks, one for each message block and one for the MIC (because of fast SRAM access, for most blocks read and write operations are done simultaneously)
- (n+1)*24 µs for the encryption of the nonces
- additional 40 µs for reading the last encrypted nonce
- about (n+1)*30 µs for bitwise XOR of encrypted nonce with message block respectively MIC.

Thus, at least ((n+1)*158 + 80) µs are required for this operation or about 1 ms for a five block message (80 bytes). This is just a lower bound. In practice, overhead for radio transmission and software (as composing the nonce, padding, assembling the frame) has to be regarded.

Figure 3-3. Estimated timing for encryption of one block in CCM mode



3.8.2 Parallelization: polling vs. waiting in CCM* mode

Since the subregister AES_DONE of register AES_STATUS can be polled to see whether the AES block has finished, encryption can be done in background, i.e. in parallel. It depends on the application whether this makes sense since sending and receiving frames are already done in background.

Parallelization of CCM* encryption requires that all needed data like AES block sequence, loop variables, nonce etc. are kept local and that the radio transceiver is neither reset nor put in sleep mode meanwhile.

For the radio transceiver, waiting 24 µs for the AES unit is normally better than polling because of minimal time win. Moreover, polling causes many SPI accesses and hence an increased power consumption.

3.8.3 Delay for decryption

An entity sending CCM* encrypted packages must regard the time which the receiver node needs to decrypt the message. This is not necessary if the frames have the same length since radio transmission is done in the background, and sending/receiving one block takes about 500 µs in 250 Kbit/s mode compared with about 130 µs for CMM* encryption (see section 3.8.1), but it does matter when a short message follows a long message.

3.9 Notes

3.9.1 Loosing key after sleep and reset

All AES registers values and in particular the key are lost if the radio transceiver goes to sleep mode (as well as on reset). So if the device periodically sleeps to save power, the key must be stored in the AVR RAM, and the AES unit must be re-initialized after each wake up.

3.9.2 SPI read after write

After a write to SPI, the first read may occur after at least 500 ns only (see [14], 12.4/12.4.11), as well as between consecutive fast SRAM accesses (i.e. write and read at the same time) to avoid undefined states.

This is important for encryption/decryption since there are many consecutive SPI operations.

4 AES register description, random number generator

4.1 AES registers

The registers and operations are described in detail in the data sheets of AT86RF231 [14], section 11.1, respectively of AT86RF212 [15], section 9.1. The following is an overview of this chapter only.

4.1.1 Register overview

How to use the registers to encrypt respectively decrypt one block has been described in detail in chapters 3.3 and 3.6.

All registers are placed in the SRAM addresses 0x82 ... 0x94:

Table 4-1. Register overview

Address	Name	Description	Subregister
0x82	AES_STATUS	AES status	AES_ER, AES_DONE
0x83	AES_CTRL	Security module control	AES_REQUEST, AES_MODE, AES_DIR

Address	Name	Description	Subregister
0x84 – 0x93	AES_KEY/ AES_STATE	Depends on AES_MODE setting: AES_MODE = 1: Contains AES_KEY (key) AES_MODE = 0 | 2: Contains AES_STATE (plain or ciphertext)	
0x94	AES_CTRL_MIRROR	Mirror of register 0x83 (AES_CTRL)	same as AES_CTRL

The AES_CTRL_MIRROR register is used to do an encryption in one SPI access only, see section 3.3.

4.1.2 Subregisters

Register AES_STATUS (0x82):

- **AES_DONE** (bit 0):

  This bit is set by the AES unit if the unit has finished its work. The bit can be used for polling.
- **AES_ER** (bit 7):

  This bit is set by the AES unit if the unit failed (e.g., after a read access to AES_CTRL while the unit is running, or after reading less than 16 byte from AES_STATE).

Register AES_CTRL: (0x83)

- **AES_DIR** (bit 3):

  This bit must be set to 1 for encryption and to 0 for decryption (see also section 3.6).
- **AES_MODE** (bits 4...6):

  Three values are allowed:

  0 - ECB mode (see section 2.2.2.1)

  2 - CBC mode (see section 2.2.2.2)

  1 - initialize key (see section 3.2)
- **AES_REQUEST** (bit 7):

  This bit must be set to 1 to start the operation.

Register AES_CTRL_MIRROR: (0x94)

This register contains the same subregisters as AES_CTRL which allows to combine all commands and data transfers required for one encryption/decryption operation in one SPI access (see section 3.3).

4.2 Hardware random number generator

The hardware random number generator (HRNG) is required for generating cryptographically secure random value, especially keys. If the radio transceiver is in basic operating mode receive state (RX_ON), the generator feeds the subregister RND_VALUE (bits 5 and 6) of the TRX register PHY_RSSI (offset 0x06) every microsecond (see data sheet [14], 11.2, and [15], 9.2).

"Cryptographically secure" means that it must be *impossible to forecast the sequence in the future*, even if members in the past are known. This is not the case for so-called pseudo random number generators (PRNGs) which are common and implemented in software. Being unpredictable is more important than having good statistical properties what can easily be reached by encrypting or hashing the values.

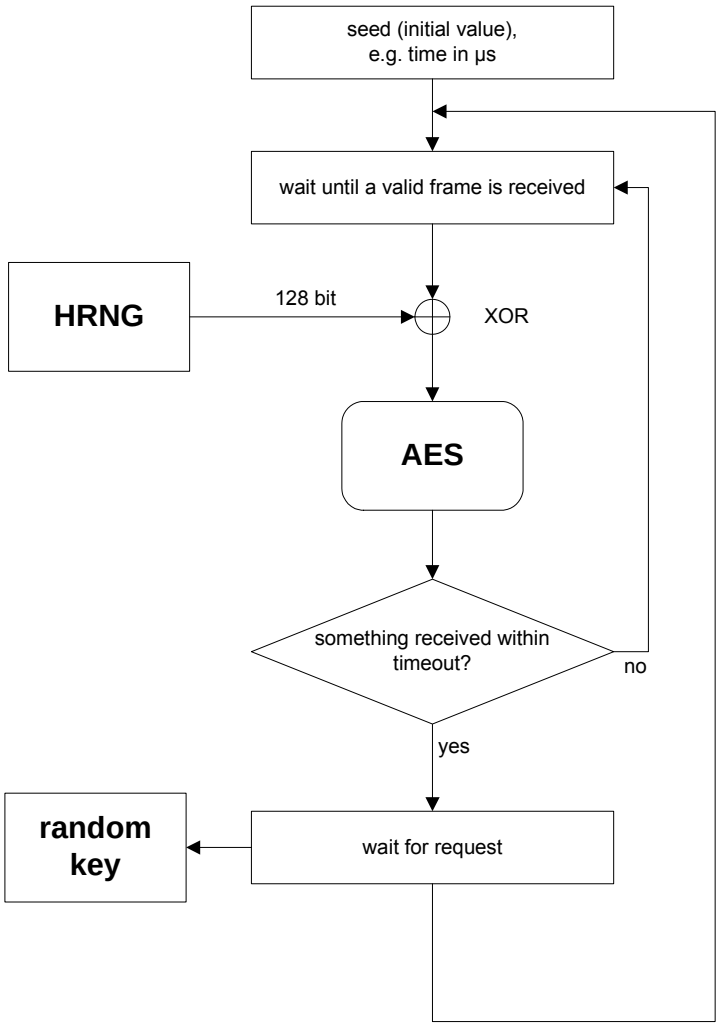
The main problem of HRNGs is to detect a malfunction. The radio transceiver contains a HRNG which does not fail as long as the radio receiver works, i.e. its malfunction can be detected as is demonstrated in the following.

Since key generation is a rare process, performance is less important. A careful approach to produce secure random keys could look as follows (see Figure 4-1):

1. Reserve some 16 byte buffer as "random pool" and initialize it with some weak random value (e.g. time in microseconds).
2. Check that the radio is in basic receive mode and perform some heuristic test that the receiver works correctly (e.g., that a frame with valid CRC was received within the last few milliseconds, no jammer is blocking).
3. Collect 128 bit from the HRNG waiting more than 1 microsecond between consecutive accesses.
4. Perform a bitwise XOR of the collected bits with the pool.
5. Encrypt the pool with the key with which the AES unit is actually initialized (the former initialization must be guaranteed by the context). Neither direction (encryption/decryption) nor encryption mode (ECB/CBC) do matter here: This step does not improve security but statistical properties of the generated keys.
6. Repeat step 2 within a short timeout (few milliseconds). If not, go to step 2.
7. Take the pool contents as actual key.

The next key generation repeats steps 2 - 7. Then the generated key is no more stored in the pool (after encrypting and distributing it).

Figure 4-1. Generating cryptographically secure random numbers in a 128 bit random pool



The additional check that some valid frame is received a short time after this procedure is a good heuristic test that the receiver worked during random number generation, i.e. that there was no malfunction. If this can not be excluded, steps 2 - 7. should be repeated. All this can also be done as a precaution in regular intervals to collect maximal entropy.

In ZigBee networks in residential mode, only the trust center has to generate keys (and only before the change of the network key, what is rare). In professional mode, keys can also be generated by devices, so the radio transceiver is also well-suited for this purpose.

5 Glossary

AES: Advanced encryption standard, one of the best and most secure encryption algorithms, see section 2.2.1.

Asymmetric encryption: → PKC

Authentication: Proof that the sender of some message is really the one who pretends to be it. Authentication is achieved by including the sender ID in the computation of a cryptographic hashsum, hence with → integrity protection.

Block cipher: An encryption algorithm where plaintext is split in blocks of equal length (128 bit in the case of →AES) and encrypted block by block, in contrast to →stream ciphers. Block ciphers are better analyzed and considered as more secure than stream ciphers, at least in the public research. Block ciphers are mostly product ciphers, i.e. very similar smaller operations (rounds) are repeated with different round keys which are generated from the encryption key.

CBC: Cipher block chaining, an →encryption mode of →block ciphers, see section 2.2.2.2. In CBC mode, identical messages do not produce identical ciphertexts, and an error in some ciphertext block causes this and all following blocks producing meaningless plaintexts after decryption.

CCM, CCM*: CCM means "counter with CBC-MAC", an encryption mode combining →CTR mode and →CBC-MAC integrity protection such that only a part of the message is encrypted but the whole message is integrity protected, see section 2.2.2.6. CCM* is CCM with optionally deactivated encryption respectively integrity protection and is used in IEEE 802.15.4 and ZigBee.

Cleartext: Unprocessed data, not compressed or encoded; in contrast to → plaintext, which is input to encryptions and needs not be cleartext.

Ciphertext: The output of an encryption.

CTR: Counter mode, an →encryption mode of →block ciphers, see section 2.2.2.4. The CTR mode is simple and secure, provided a unique →nonce is used and →integrity protection is applied.

DES: Data encryption standard, the old default encryption standard from 1977. The key length of 56 bit is now too short to withstand modern hardware. The algorithm was replaced by → AES in 2000.

ECB: Electronic code book mode, the simplest →encryption mode of →block ciphers where each plaintext block is separately encrypted, independent of other blocks. It is unsecure for longer messages; in particular, it produces identical ciphertexts for identical plaintexts. ECB mode is usually applied to encrypt random session keys. See section 2.2.2.1.

Encryption mode: The way a →block algorithm is applied, e.g. whether each block is encrypted straightforward, or with the help of other data. Examples here: →ECB, →CBC, →CTR.

Frame counter: A frame number contained in the header which must be unique at least as long the same key is applied. In ZigBee, frame counters are 32 bit long and start with 0. Frame counters protect against →replay attacks.

HRNG: Hardware random number generator, a device generating random using physical sources. In contrast to →PRNGs, they should yield cryptographically secure random which cannot be guessed knowing the past numbers. In high security applications, HRNGs are often mixed with PRNGs.

Integrity protection: A cryptographic protocol that allows to detect unauthorized modification of a message. Integrity protection is often incorrectly called → authentication.

Key initialization: Before encryption keys are used, they are often transformed. E.g., for product ciphers (→block ciphers), round keys are generated from the key. This is

called key initialization. In the AES unit of the radio transceiver, key initialization is done internally during encryption.

MAC: Message authentication code, a cryptographic hashsum providing →integrity protection. To compute a MAC or to check its correctness, a secret key must be known (in contrast to digital signatures).

MIC: Message integrity code, the →MAC used in IEEE 802.15.4 and ZigBee. According to Wikipedia [17], MICs can be computed without secret keys and have to be encrypted. So the ZigBee MIC is essentially an encrypted MAC.

Nonce: Derived from *number used once*, in the context of this note an AES block which is never repeated and used for CBC-MAC computation and CTR encryption. The IEEE 802.15.4/ZigBee nonce contains →frame counter, sender ID, key number and other parameters.

Padding: To apply a block cipher, plain respectively ciphertext must have lengths which are multiples of the block length. This must be done by padding, where bytes (usually 0) are appended at the plaintext. There are several methods to do this, see [4], 9.1 and 9.3, and [5], 5.1.2. In IEEE 802.15.4, header and payload are padded separately; the lengths can be seen from the header.

PKC: Public key cryptography which uses a publicly known key to encrypt a message but requires a secret (private) key to decrypt it. It is used for secure transmission of random session keys used for →symmetric key algorithms and for digital signatures. In contrast so symmetric key cryptography, PKC is hard to implement and requires much computing power respectively special hardware.

Plaintext: An unencrypted text.

PRNG: Pseudo random number generators.

Product cipher: →block cipher

Replay attack: An attacker sends an already sent message

Round key: →block cipher

RNG: Random number generator, see →HRNG and →PRNG

Stream cipher: In contrast to →block ciphers, a key-dependent bit stream is generated. Encryption and decryption are the same operation: bitwise XOR of key stream with the plaintext. Stream ciphers require →integrity protection because of bit flip attacks.

Symmetric key algorithm: The "usual" cryptographic algorithm, where the same secret key is used for encrypt and decryption.

6 References

[1] www.nist.gov/aes (AES development, historical site)

[2] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (AES standard and test vectors)

[3] http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html (overview over block ciphers, key handling and test vectors)

[4] Schneier, B.: Applied Cryptography, 2nd ed., Wiley1996

[5] Wobst, R.: Cryptology Unlocked, Wiley 2007

[6] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

[7] http://en.wikipedia.org/wiki/CCM_mode

[8] http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf

[9] http://tools.ietf.org/html/rfc3610

[10] http://en.wikipedia.org/wiki/EAX_mode

[11] IEEE 802.15.4 standard: http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf

[12] ZigBee standard: http://www.zigbee.org

[13] http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/CCM/RW_CCM_comments.pdf

[14] AT86RF231 data sheet: http://www.atmel.com/dyn/resources/prod_documents/doc8111.pdf,

[15] AT86RF212 data sheet: http://www.atmel.com/dyn/resources/prod_documents/doc8168.pdf

[16] http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html

[17] http://en.wikipedia.org/wiki/Message_authentication_code

[18] 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

7 Table of Contents

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature