Atmel

APPLICATION NOTE

AVR411: Secure Rolling Code Algorithm for Wireless Link

AVR411

Features

- Uses advanced encryption standard (AES) and its cipher-based message authentication code (CMAC) mode of operation for transmitter authentication
- 128, 192 or 256 Bit key sizes supported
- Uses sleep modes for low-power consumption
- Less than 30ms response time
- PC command-line tools for cryptographic key management
- Multiple transmitters supported
- Secure learning mechanism for introducing new transmitters

Introduction

This application note describes a Secure Rolling Code Algorithm transmission protocol for use in a unidirectional wireless communication system. Typical applications for this algorithm are garage door openers, remote keyless entry, passive entry, and remote car start systems.

The transmission protocol can be used in a system consisting of one receiver and a limited number of associated transmitters. Two critical qualities of this protocol are:

(1) No transmission is ever repeated which prevents a would-be thief from grabbing the message and retransmitting and

(2) It is virtually impossible to predict message contents, even if previous messages are known. The receiver ignores all messages that have already been used.

A working implementation written in C is included with this application note. Full documentation of the source code and compilation information is found by opening the *readme.html* file included with the source code.

2600E-AVR-07/15

Figure 1. Secure Rolling Code Development Tools (STK512)

This application note also describes how to use the Secure Rolling Code Starter Kit (STK512) from Atmel as a platform to evaluate this algorithm in a simple RF wireless control system. In addition to STK512, an ATtiny44 microcontroller is used in the transmitter and an ATmega88 in the receiver.

1. Overview of Concept

The following goals can be set up for the final system:

1. One receiver should be associated with a limited number of transmitters
2. No two transmitted messages should be equal
3. The receiver should ignore messages that have already been used
4. It should be virtually impossible to predict the next message contents from a transmitter, even if previous message contents are known

The motivation for the first goal is obvious. In the case where the system is used with a car, one would not want all keys from the car manufacturer to fit each and every car ever produced. Instead, each car owner gets a unique set of keys that only fit his/her car.

The second goal prevents a would-be thief from grabbing a transmitted message and retransmitting it at a later time. If all messages were equal, there would be no security in the system. Stealing a car would be easy.

The third goal follows naturally from the second goal. Since there is no communication back to the transmitter, the receiver must keep track of which messages that have already been used and ignore them if retransmitted.

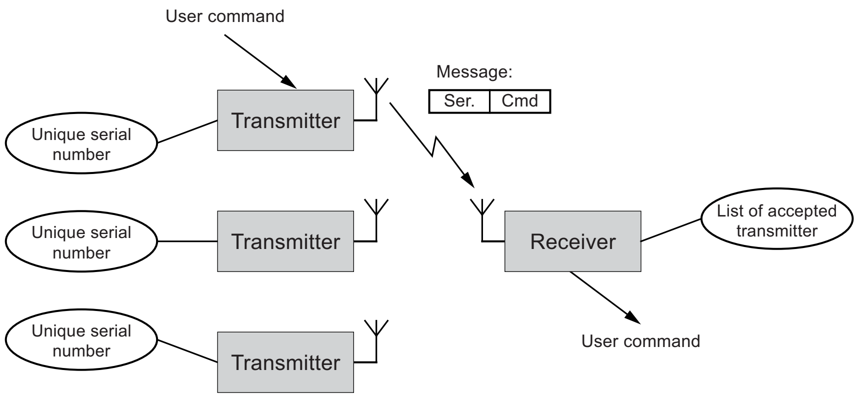
Even with every message being different, there has to be some sequence known by both the transmitter and the receiver. This is due to the unidirectional communication link. With some effort, a would-be thief could grab a number of messages and use that information to predict the contents of the next message. Next, using the predicted message, the system is easily broken. Therefore an effort must be made to make it virtually impossible to predict future messages, hence the fourth goal.

2. Theory of Operation

2.1 System Overview

A basic system without any security is shown in Figure 2-1 below. It shows three transmitters, fobs, and one receiver. The requirement that a receiver is associated with a limited number of transmitters is satisfied using unique serial numbers, or IDs, for the transmitters. A list of accepted transmitter IDs is maintained in the receiver. How the list of accepted transmitters is maintained and updated is described (see Section 2.2 "Learning New Transmitters" on page 7).

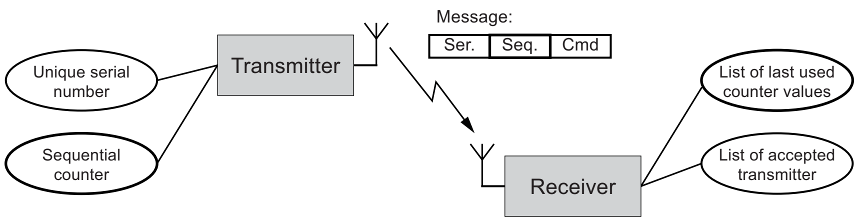
Figure 2-1. Basic System with Goal 1 Satisfied



The requirement that each message should have different contents, even if the same command is sent twice, is easily achieved by including a sequential counter in the message. This counter is incremented after each message transmission.

To satisfy the requirement that the receiver should ignore messages that have already been sent, a certain amount of synchronization between the transmitters and the receiver is required. Taken into account that the communication goes one way only, an obvious solution is that the receiver keeps track of the last used sequential number for each transmitter. The receiver then simply ignores messages having old sequential numbers. Attempts to retransmit old messages are therefore, useless. The upgraded system, meeting goals 1, 2 and 3, is shown in Figure 2-2 on page 4.

Figure 2-2. Upgraded System with Goals 1, 2 and 3 Satisfied

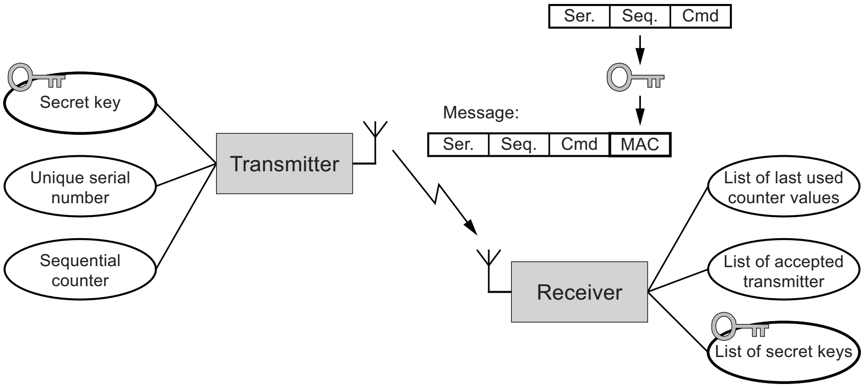


The solution described above does not satisfy the fourth goal, though. A person having the contents of a few consecutive messages can easily predict the next message content. What we need is to include an element in the message that is too large to guess by trial-and-error and virtually impossible to predict without knowing a secret key which is unique to each transmitter. Since each message should be different, the extra element should be dependent on the secret key and the message contents itself. Including such an element satisfies the message unpredictability requirement.

An element that is dependent on a secret key and the message contents is commonly referred to as a Message Authentication Code, or MAC for short. When using a MAC, it is trivial to predict the next sequential number, but impossible to provide a valid MAC without knowing the secret key. Since the MAC is transmitted with the rest of the message, it is therefore impossible to predict the complete message. Figure 2-3 shows the system using a MAC to authenticate the message.

There are numerous algorithms available to use for generating the MAC, but for various reasons we have chosen the Advanced Encryption Standard (AES) algorithm, which is a symmetric block cipher. The AES algorithm supports key sizes of 128, 192 and 256 bits. Its use as a MAC generator is discussed further under theheading Section 2.1.1 “Rolling Windows” on page 5.

Figure 2-3. Secure System with all Four Goals Satisfied

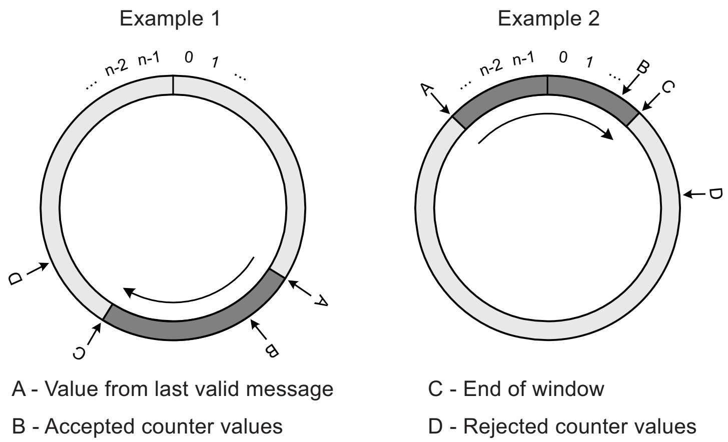


2.1.1 Rolling Windows

The concept of simply ignoring messages having old sequential numbers leaves one problem: What if the counter value overflows and wraps back to 0? This section describes a solution.

Handling the sequential counter is best described by two examples, given in Figure 2-4. The first example shows a situation where the last received valid message had a counter value A. As there is always the possibility that the transmitter has been activated a number of times outside the receiver's range, the receiver must accept values up to some limit, labeled C in the figure. The simple approach of accepting all values larger than the last received value won't work, as is apparent in the second example where point A is close to the upper end of the counter value range. The dark segment from point A to C shows the window of acceptance for counter values. Point B is an example of a value that would be accepted while point D is a value that would be rejected. When a value is accepted, the window starting point moves to that point.

Figure 2-4. Rolling Window of Acceptance for Counter Values



This scheme ensures that old messages are never accepted unless the head of the rolling window has reached the old counter values. By choosing a large enough counter span and limiting the window size itself, this scheme effectively prevents replay attacks with old messages.

2.1.2 Message Format

When designing the message format, the following must be considered:

- Data rate of the communication link (STK512)
- Power consumption
- Uniqueness of the transmitter serial number
- Rolling window size
- Security requirements for the MAC
- Required command range

To simplify the implementation, all fields should be a multiple of 8 bits.

The serial number should be large enough to ensure that no two transmitters have equal number. Of course, if two transmitters have equal serial numbers, their secret keys would be different, and thus the receiver would regard the MAC as being invalid. However, such a situation would seem like an attack attempt to the receiver, which is not a preferred solution.

It must also be taken into account the probability of two transmitters with equal serial numbers be used within each other's range. This is very application-specific, but a serial number of 32 bits is large enough for most cases. If power consumption is a major issue, the serial number could be reduced to 16 bits. Less than 16 bits is not recommended unless the total transmitter count is less than 216 = 65536.

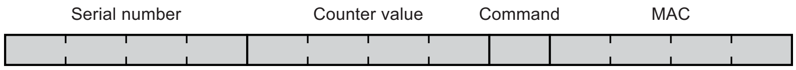
The command ranges for the intended systems are typically very limited and 8 bits for the command field is enough for most applications.

When choosing the size for the sequential counter, one must consider the rolling window size. The counter range should be at least twice the window size, preferably larger. The designer must estimate the probability of the transmitter being activated outside the receiver's range. This is also very application-specific, but 32 bits will cover all but the most exceptional cases. If power consumption is a major issue, 16 bits could be used after careful analysis of the application. Less than 16 bits is not recommended.

The MAC size depends on security requirements. According to [2], the MAC should be at least half the size of the secret key, but could be a small as 32 bits if numerous trials are impractical due to bandwidth limitations in the system. This is the case for our system and we choose 32 bits for the MAC field.

From the discussion above, the following message format is suggested: 4 bytes for the serial number, 1 command byte, 4 bytes for the counter value and 4 bytes for the MAC. That is a total of 13 bytes or 104 bits. This format will be used in the rest of this application note, but is easily customized.

Figure 2-5. Suggested Message Format



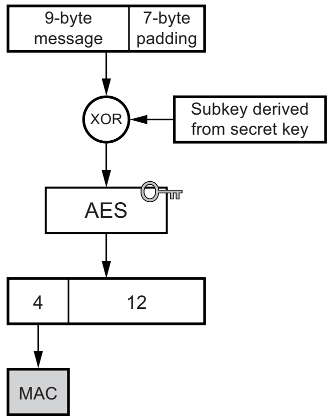
2.1.3 Generating the MAC

The details of the AES algorithm itself are outside the scope of this document. The interested reader is referred to [3] for the original AES specification and [5] for an AVR® implementation of the algorithm. This section covers the use of AES as a MAC generator.

The motivation for choosing a cipher-based MAC instead of a hash-based, like SHA-1, is that a block cipher algorithm is a powerful feature that could be useful in other parts of the application.

The Cipher-based MAC (CMAC) mode of operation for the AES algorithm is used to generate the MAC. The details of this mode are found in [6] and [7]. The short messages used in this application reduce the complexity of the MAC generation considerably compared to a full implementation of the AES-CMAC. The operation is illustrated in Figure 2-6 on page 7. The 9-byte message (excluding the MAC) is first extended to 16 bytes by padding with a single '1'-bit and 55 '0'-bits. The 16-byte block is XORed with an equally large subkey. The result is then encrypted using the secret key and the upper 4 bytes of the ciphertext are used as the MAC.

Figure 2-6. CMAC Mode for AES



The subkey is derived by modular multiplication of a 16-byte block L by 4, where L is a 16-byte block of '0'-bytes encrypted using the secret key. The modulus is an irreducible binary polynomial, R128, represented by one '1'-bit followed by 120 '0'-bits and the binary string '10000111'. If the message format is changed to a total of 16 bytes, no padding is required and the modular multiplication factor is 2 instead of 4. Please refer to [6] and [7] for more details.

2.2 Learning New Transmitters

This section describes the mechanisms behind introducing new transmitters into the system. It includes both an overview of the algorithms that are used and the recommended procedures for handling the cryptographic keys. The previous sections assumed that the receiver already had an updated list of the secret keys for all accepted transmitters. Even if the receiver was preprogrammed with this information, which it is not, you need a method of adding transmitters to the receiver's list without compromising the security of the system. This is the goal of this section.

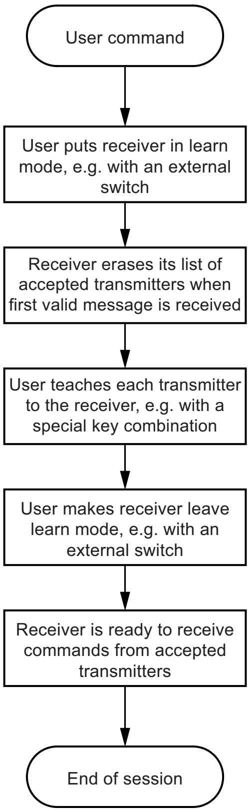
2.2.1 Key Transfer

As discussed under heading Section 2.1 “System Overview” on page 4, the transmitters are preprogrammed with a unique identification number and a secret encryption key. To be able to recognize and accept a new transmitter, the secret key must be transferred to the receiver in a secure manner. It is important that the transmission from the transmitter is unreadable for all but the receivers belonging to the same system. It is also important that the receiver only accepts transmitters belonging to the same system. To achieve this security, we introduce a cryptographic key shared by all transmitters and receivers belonging to one system.

It is up to the designer to select the scope of this shared key. For instance, the shared key could be unique for every manufactured car, which means that transmitters must be preprogrammed with the shared key for one particular car before shipping to the customer. A more practical solution is to have one shared key for all cars of one particular model, which means that transmitters must be preprogrammed with the shared key for the car model in question. Choosing a narrower scope for the shared key reduces the number of units having the same shared key, thereby increasing the cryptographic security of the system.

Figure 2-7 on page 8 shows the flow of a typical learning session.

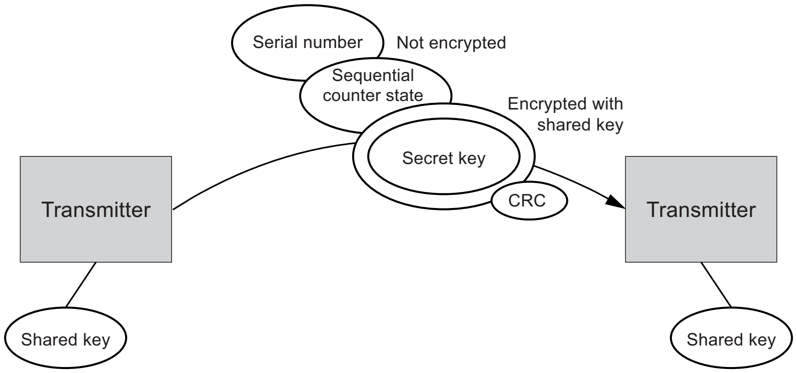
Figure 2-7. The Flow of a Learning Session



The key transfer process is initiated by putting the receiver into a special learn mode using an external signal, e.g., a switch hidden inside the car. On the transmitter side, the learning process is initiated by e.g., a special key combination. The transmitter encrypts its secret key using the shared key and transmits it to the receiver together with the serial number and optionally the sequential counter state. Due to this scheme, the shared key is often referred to as a key encryption key (KEK).

The details of the key transfer are illustrated in Figure 2-8.

Figure 2-8. Secret Key Transfer Session



If one can assume that the transmitter is not operated before it is used in a key transfer session, the counter state can be omitted from the transmission, resulting in a shorter transmission. Even if the transmitter has been operated a few times, it will be accepted as long as the counter has not counted past the rolling window size. The implementation in this application note includes the counter value in the transmission.

To ensure message integrity, a CRC code generated from the whole message frame is also appended to the message. This is described in more detail in the implementation section.

2.2.2 Where to Store the Shared Key

Although the shared key used in the learning process is referred to as 'shared', that does not mean that it should not be kept secret. The car manufacturer would have a database that pairs every system with its shared key. System security depends on keeping this database secret. Also note that the secret key unique to every transmitter should not be stored anywhere except inside the transmitter itself.

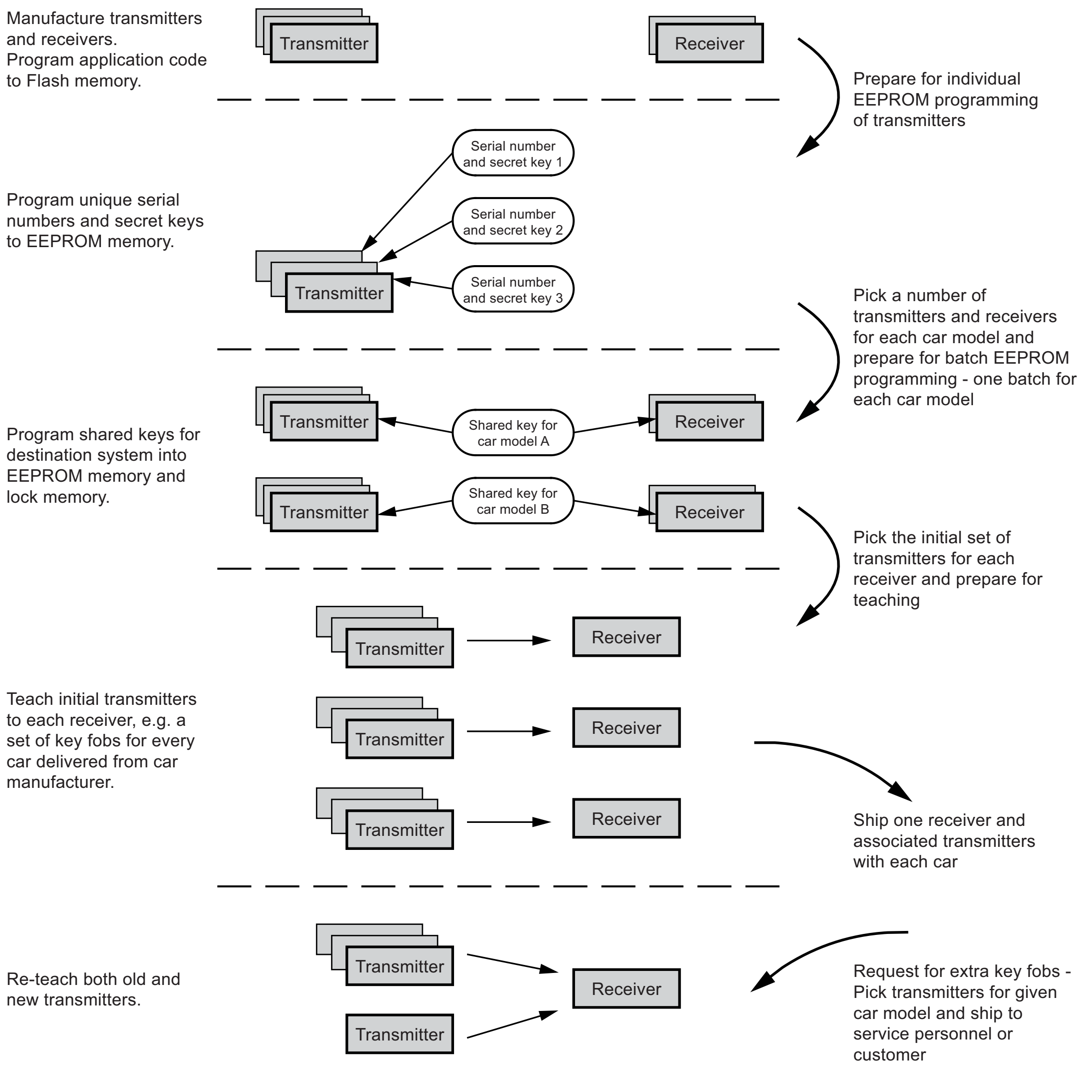
2.2.3 Safety Against Unauthorized Key Introductions

The car owner wants to make sure that no keys are introduced to the system without him/her knowing it. The service personnel could hypothetically introduce keys when car is at service. To eliminate the possibility that keys are introduced without the owner knowing it, this implementation cannot simply add a transmitter to the system. Instead, when the receiver gets an external signal from e.g. a switch on the dashboard, the entire list of accepted transmitters is erased and the receiver is ready to learn the serial number and cryptographic keys of one transmitter at a time.

2.2.4 From Production to End User

Figure 2-9 follows the transmitters and receivers from production through programming and installation, to shipping to the end user. It also shows the introduction of replacement or extra transmitters.

Figure 2-9. Overview of Transmitter and Receiver Life



As the figure shows, the requirement that both old and new transmitters are taught to the receiver ensures that no transmitters are introduced to the system without the owner knowing it. For instance, the car owner would notice that his own spare key fob he or she left at home no longer works when the car is returned from service, if someone tried to teach another transmitter to the car's receiver without the owner's permission.

3. Implementation

An overview of the hardware implementation is shown in Figure 3-1. It shows the development platforms for the receiver and the transmitter.

The transmitter is implemented as a key fob with three push buttons, one LED and a 6-way connector for ISP programming and debugging using the debugWIRE interface of the ATtiny44.

The receiver platform consists of the STK®500 board with the RF receiver top module. The communication between the ATmega88 on the STK500 and the RF receiver top module goes through the expansion bus connectors EXPAND0 and EXPAND1. Note that the DATA-switch on the top module must be put in the position marked "STK500".

Switch SW5 is used to enter learn mode, with visual feedback on LED5. LED1-4 provide a visual response to received and accepted commands. 10-way ribbon cables connect the switches to PORTD and the LEDs to PORTC.

Figure 3-1. Hardware Overview

3.1 The Transmitter

3.1.1 Hardware

Figure 3-2 shows the connections between the key fob switches, LEDs, ATtiny44 and the RF transmitter. The device used in this application consists of three switch inputs. The LED glows dimly when the switch is pressed. In addition, the PA6 pin is also used for controlling the amplitude modulation of the RF transmitter.

Figure 3-2. Transmitter Hardware Implementation

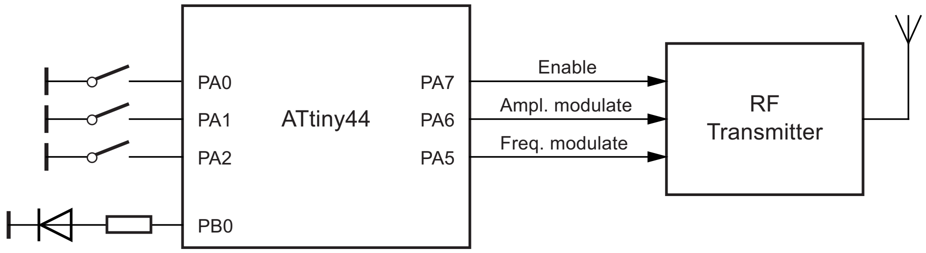
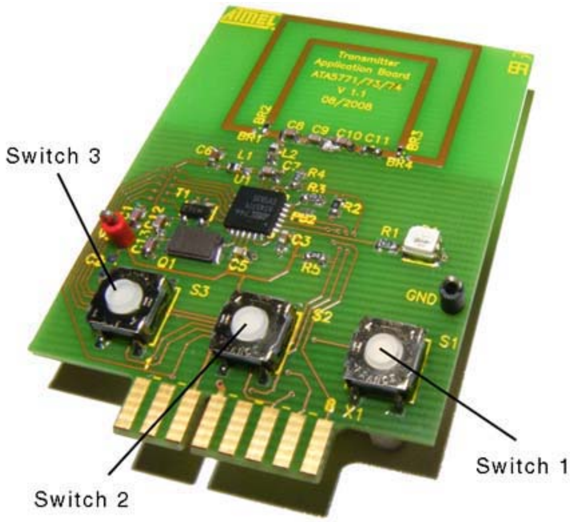


Figure 3-3. Transmitter Button Placement



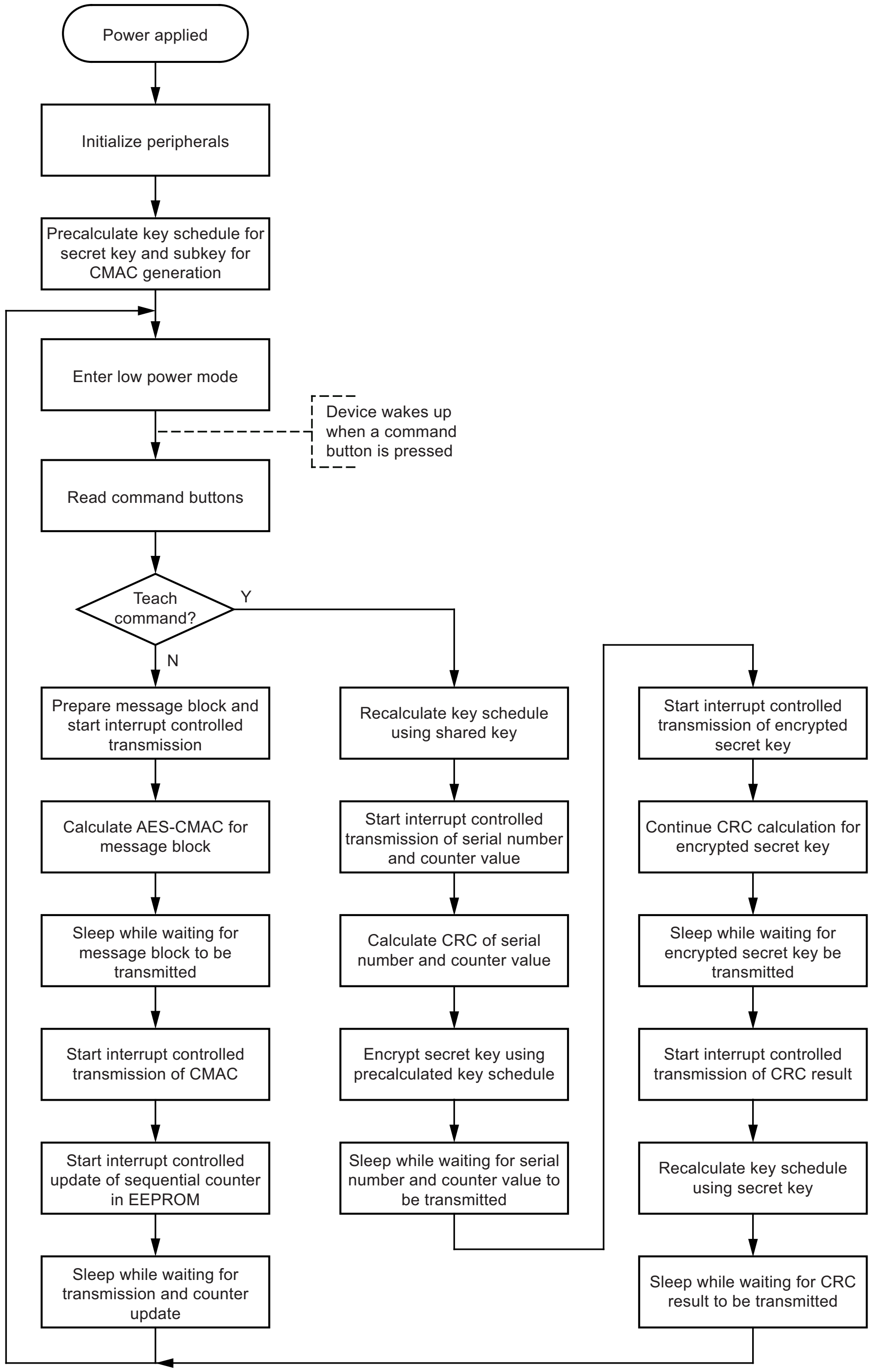
3.1.2 Software

The transmitter spends most of its time in Power Down Sleep Mode, waiting for the user to press a button. When a Pin-Change Interrupt wakes up the ATtiny44, the button selection is compared to a defined Teach Command button. If it matches, the transmitter encrypts and transmits its secret key using the system's shared key. If it is an ordinary command, e.g., Unlock Car, the transmitter constructs a message with a MAC and transmits it using the USI module and Timer/Counter0 to control the RF transmitter device. Before entering Power Down Sleep Mode again, the sequential counter value is incremented. A detailed flowchart is shown in Figure 3-4 on page 13.

For this application example, the Teach Command is defined as Button 3.

Note: The extensive use of interrupt-controlled operations, e.g., transmission, to be able to spend as much time as possible in Idle Sleep Mode, thereby reducing power consumption. Deeper sleep modes are not possible during operation, since the interrupts used cannot wake up the device from other modes than Idle Sleep Mode.

Figure 3-4. Flowchart for Transmitter Operation



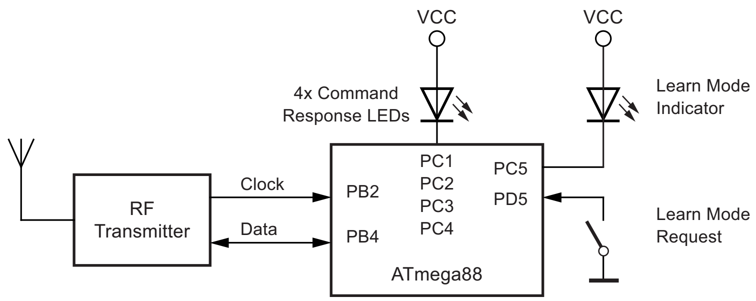
3.2 The Receiver

3.2.1 Hardware

Figure 3-5 shows the connections between the RF receiver, ATmega88, Learn Mode LED and switch and the command response LEDs. The I/O-pins are chosen this way to simplify implementation and prototyping on STK500 and the RF receiver top module.

Note: Only two signals are required to communicate with the RF receiver. This leaves lots of I/O free for customization of the application.

Figure 3-5. Receiver Hardware Implementation



3.2.2 Software

The receiver too spends most of its time in Power Down Sleep Mode, waiting for incoming messages or a request for entering learn mode.

When a valid RF signal is received, the RF receiver wakes up the ATmega88 with a low clock pulse. The receiver then reads the data bits on each successive clock pulse until either a whole message has been received or the reception times out. The ATmega88 enters Idle Sleep Mode between each clock pulse and wakes up from a clock pulse or a timeout condition. The clock pulse triggers a Pin-change Interrupt and the timeout is implemented using Timer/Counter1 Compare Match Interrupt A.

If a press on the Learn Mode Request switch wakes up the device, the software enters Learn Mode and waits for any transmitter to transmit its teach message. Using Timer/Counter1 Compare Match Interrupt B to implement a 10 second delay, the user have some time to press the correct button combination on the individual transmitters.

The receiver operation is shown in Figure 3-6, Figure 3-7 on page 15 and Figure 3-8 on page 16 below.

Figure 3-6. State Diagram for Receiver Operation

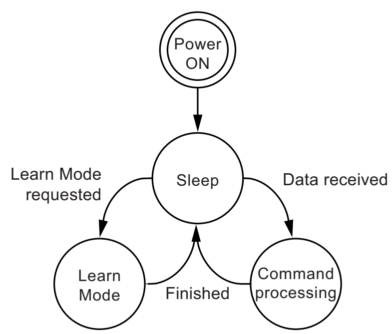


Figure 3-7. Flowchart for Command Reception and Processing

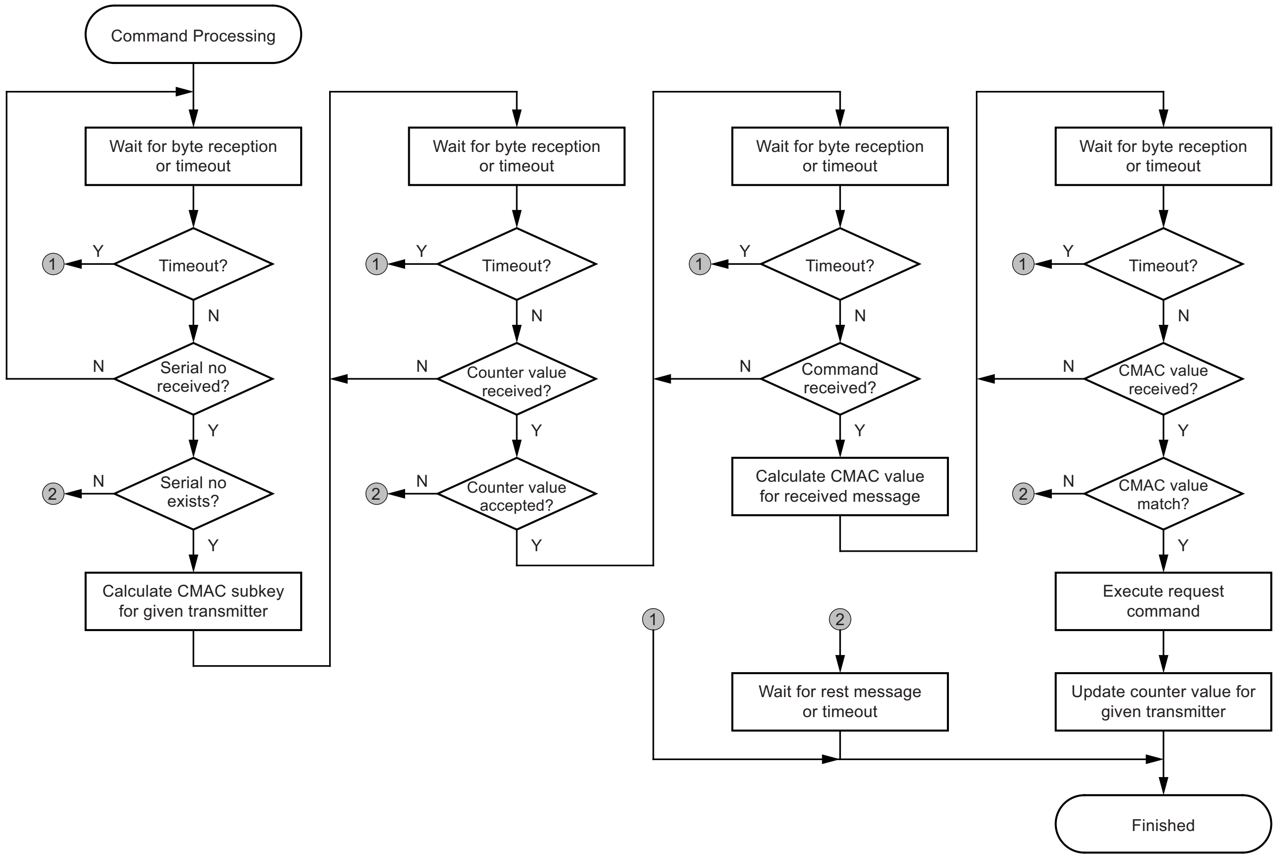
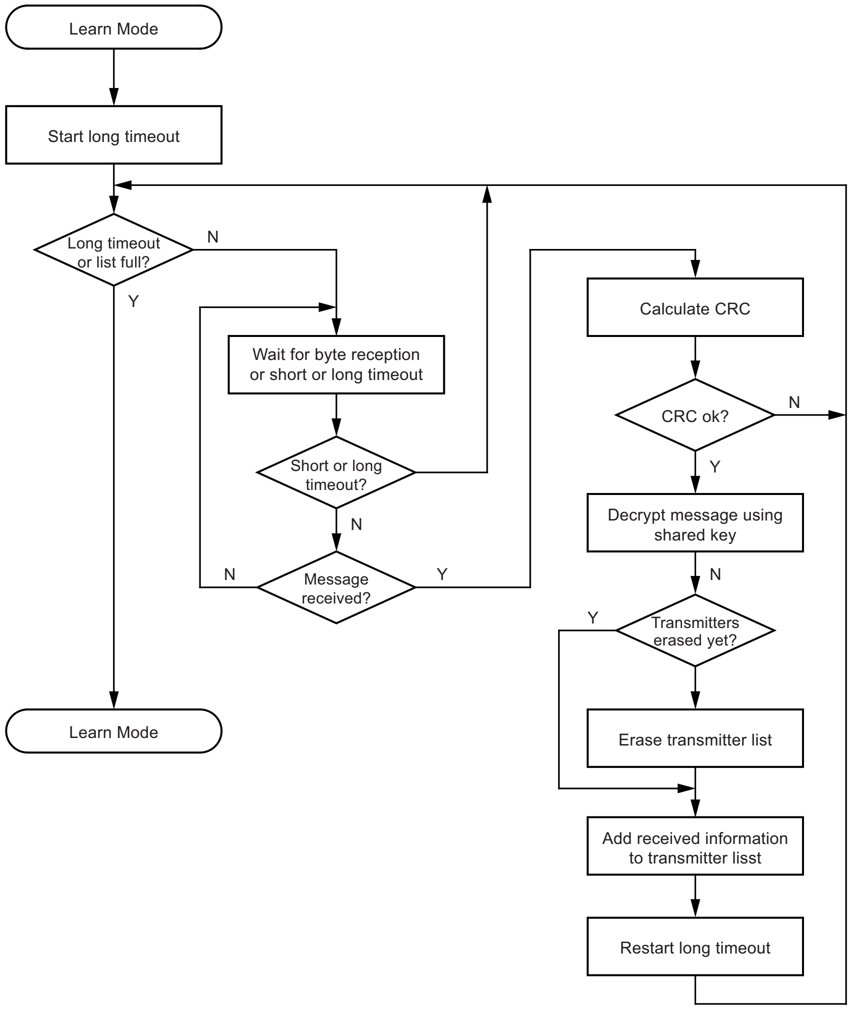


Figure 3-8. Flowchart for Receiver in Learn Mode



3.3 4.3 Command-line Tools for PC

Two command-line tools are included with this application note. Full source code and binary executables for Win32® and Linux® platforms are included. The tools are used to generate HEX files containing the initialization data to be programmed into the EEPROM memory of transmitters and receivers.

3.3.1 Transmitter data

The command line syntax for this application:

```
createtxhex filename keysize serialnosize countersize sharedkey secretkey serialno
countervalue
```

An example that creates a HEX file for a transmitter using 128-bit AES, 4 byte serial numbers and 4 byte counters:

```
createtxhex tx00002A3C.hex 128 4 4 00112233445566778899AABBCCDDEEFF
0123456789ABCDEF0123456789ABCDEF 00002A3C 00000001
```

Table 3-1. createtxhex Parameters

Parameter Name	Description
filename	Filename for the resulting Intel HEX file
keysize	Size in bits of the AES secret and shared keys (128, 192 or 256)
serialnosize	Size in bytes of the serial number (1, 2 or 4)
countersize	Size in bytes of the sequential counter (1, 2 or 4)
sharedkey	Hex representation of the shared key (32, 48 or 64 digits)
secretkey	Hex representation of the secret key (32, 48 or 64 digits)
serialno	Hex representation of the serial number (2, 4 or 8 digits)
countervalue	Hex representation of current counter value (2, 4 or 8 digits)

3.3.2 Receiver Data

The command line syntax for this application:

```
createrxhex filename keysize serialnosize countersize sharedkey maxtransmitters
```

An example that creates a HEX file for a receiver using 128-bit AES, 4 byte serial numbers and 4 byte counters and can have no more than 5 transmitters associated with it:

```
createrxhex rx0001.hex 128 4 4 00112233445566778899AABBCCDDEEFF 5
```

Table 3-2. createtxhex Parameters

Parameter Name	Description
filename	Filename for the resulting Intel HEX file
keysize	Size in bits of the AES secret and shared keys (128, 192 or 256)
serialnosize	Size in bytes of the serial number (1, 2 or 4)
countersize	Size in bytes of the sequential counter (1, 2 or 4)
sharedkey	Hex representation of the shared key (32, 48 or 64 digits)
maxtransmitters	Maximum number of associated transmitters

4. Code Size and Performance

When running without the Brown-out Detector enabled, the transmitter kit (ATtiny44 and RF transmitter) running at 8MHz internal RC consumes approximately 0.3A on average. Using a 3V Li-cell with a typical capacity of at least 200mAh, the transmitter will operate for at least 75 years. The battery self-discharge rate will be the limiting factor.

The RF receiver has a maximum data rate of 10Kbps. Using Frequency Shift Keying and Manchester encoded transmissions, stable operation is achievable at 4800baud/s, giving a system reaction time of less than 30ms with a message size of 13 bytes and the RF receiver wake-up delay.

The code sizes are approximately 5.7Kbytes for the receiver's ATmega88 and approximately 2.4Kbytes for the transmitter's ATtiny44. The IAR Embedded Workbench® AVR® C-Compiler 4.11A has been used with available optimization.

5. Potential Improvements

If an attacker intercepts a message without the receiver getting it, a successful replay attack is possible if the receiver does not receive more messages from the transmitter first. The rolling window scheme only provides limited security against such attacks, as the period of vulnerability is limited from the time of interception until the next received transmission. One way of securing the system against such attacks is to have synchronized clocks in the transmitters and the receiver. This would consume more power.

It should be noted that no cryptographic system is more secure than its weakest link. In this case that means that even if the AES algorithm and its use as a MAC generator are considered secure, it is still the user's own responsibility to develop and implement a key management infrastructure. This includes manual procedures, access control to sensitive information, backup and destruction routines and so on.

6. Quickstart Guide

This section contains the necessary initial steps to get a simple system with one receiver and a few transmitters up and running quickly. You should have the following ready before you start:

- At least one key fob with ATtiny44
- An STK500 board with an ATmega88 in socket SCKT3200A2
- The STK512 top module and RF receiver board mounted on the STK500
- Two 10-way ribbon cables to connect LEDs and switches
- Latest release of AVR Studio® 4

The following is optional:

- JTAGICE mkII if you want debugging possibilities
- IAR Embedded Workbench AVR® C compiler if you want to change and recompile the source code without porting it to another compiler (precompiled source code with default configuration is provided)

6.1 Configure Options

This step can be skipped if you want to use the precompiled source code with default settings.

There are numerous options for the system, e.g., cryptographic key sizes, message field sizes etc. The parameters are given as #define macros in the config.h files in both the transmitter and receiver source code folder. The most important parameters are given in Table 6-1. It is important that the parameters for the transmitter and receiver code are equal.

The configuration files contains several other parameters, but those are more advanced options. These parameters' usage is explained in comment blocks in the configuration files themselves, and should not be altered unless you know what you are doing. Always keep a backup copy of the original default configuration.

Table 6-1. Basic Configuration Parameters

Parameter Name	Default Value	Description
KEY_BITS	128	Size of the AES cipher key in bits. Allowed values are 128, 192 and 256 bits, where 256 bits is the most secure option.
SERIAL_NO_BYTES	4	Size in bytes of the message field containing a transmitter's serial number. Allowed values are 1, 2 and 4 bytes. (1)
COMMAND_CODE_BYTES	1	Size in bytes of the message field containing the requested command. Allowed values are 1, 2 and 4 bytes. (1)
SEQ_COUNTER_BYTES	4	Size in byte of the message field containing the sequential counter value. Allowed values are 1, 2 and 4 bytes. (1)
MAC_BYTES	4	Size in bytes of the message field containing the MAC. The value must not be larger than 16 bytes. More bytes gives a more secure authentication.
MAX_TRANSMITTERS	5	Maximum number of transmitters that one receiver can learn. This number is limited by the amount of free EEPROM memory. A compile error will occur of the number is chosen too large.
WINDOW_SIZE	100	The size of the rolling window of acceptance. See Section (see Section 2.1.1 "Rolling Windows" on page 5).

Note: 1. Serial number, command code and sequential counter value fields must not exceed a total of 16 bytes. A compile error will occur if the total size exceeds this limit.

6.2 Compile Projects

This step can be skipped if you want to use the precompiled source code with default settings. If not, compile projects for both the transmitter and the receiver. Detailed compilation instructions and fuse settings are giving in the source code documentation.

6.3 Generate EEPROM Images

Allocate serial numbers and secret and shared keys for the system components. Then use the supplied command line tools to generate one HEX file for every unit. The secret key for the transmitters should be discarded after generating the HEX file. They are not needed and could compromise system security if they get into wrong hands.

Make sure that all transmitters that are to be associated with a receiver have the same shared key as the receiver.

Note: The supplied tools are only meant for prototyping and evaluation. For full production use, a secure key management infrastructure must be established.

6.4 Program Flash and EEPROM Memories

At this point you should have the following HEX files ready:

- Software for the transmitters
- oftware for the receiver
- EEPROM data for each transmitter
- EEPROM data for the receiver

Use any tool capable of ISP programming, e.g., STK500, JTAGICEmkII or AVRISP, to program the ATmega88 in the receiver and the ATtiny44 in the transmitters. The devices should be shipped with ISP enabled by default.

6.5 Teach Transmitters to Receiver

Enter learn mode on the receiver by pressing the SW5 button on the STK500. The LED marked LED5 lights up, indicating that the receiver is ready. Keep your key fobs ready. The default timeout for learn mode is 10s.

Within 10s, press the teach-button on one of the key fobs you want to teach to the receiver. The learn-mode LED blinks off once, and you have another 10 seconds to teach the next transmitter. If the LED does not blink, the message was not received correctly, perhaps due to RF noise or the correct button combination was not used. Just try again until the LED blinks once.

When the last transmitter has been taught to the receiver, wait for the 10-second learn-mode timeout to expire, LED5 goes off and the receiver is ready to accept messages from the transmitters.

If the maximum number of transmitters is reached, the learn-mode LED blinks to indicate that a transmitter has been learned, but there is not another 10-second delay. The LED goes off immediately after blinking.

6.6 Ready to go!

Now the system is ready for operation. Press a button or a combination of buttons on a key fob and watch the STK500 LEDs change accordingly.

If you want to experiment with the receiver's command responses, take a look at the `ExecuteCommand(...)` function in the main.c file. If you change the function body, keep its execution time as short as possible to prevent the watchdog from timing out. The watchdog timeout delay is set to approximately 8 seconds by default.

Enjoy!

7. References

1. FIPS Publication 180-2, Secure Hash Standard (SHS), U.S. DoC/NIST, August 1, 2002.
   http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
2. FIPS Publication 198, The Keyed-Hash Message Authentication Code (HMAC), U.S. DoC/NIST, March 6, 2002.
   http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf
3. FIPS Publication 197, Advanced Encryption Standard (AES), U.S. DoC/NIST, November 26, 2001.
   http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
4. Joachim Lechner, Markus Tatzgern: Efficient implementation of the AES encryption algorithm for Smart-Cards, June 6, 2004.
   http://www.iaik.tu-graz.ac.at/teaching/10_seminare-projekte/01_Telematik%20Bakkalaureat/EfficientAESImplemetation.pdf
5. Application Note AVR231: AES Bootloader, Atmel Corporation, April 2005.
   http://www.atmel.com/dyn/resources/prod_documents/doc2589.pdf
6. Tetsu Iwata, Kaoru Kurosawa: One-Key CBC MAC, December 20, 2002.
   http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/omac/omac-spec.pdf
7. NIST Special Publication 800-38B, The CMAC Mode for Authentication, U.S. DoC/NIST, May 2005.
   http://csrc.nist.gov/CryptoToolkit/modes/800-38_Series_Publications/SP800-38B.pdf
8. Alfred J. Menenez, Paul C. Van Oorschot, Scott A. Vanstone: Handbook of Applied Cryptography, CRC Press, August 2001.
   http://www.cacr.math.uwaterloo.ca/hac/

8. Revision History

Please note that the following page numbers referred to in this section refer to the specific revision mentioned, not to this document.

Revision No.	History
2600E-AVR-07/15	• Put document in the latest template

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**