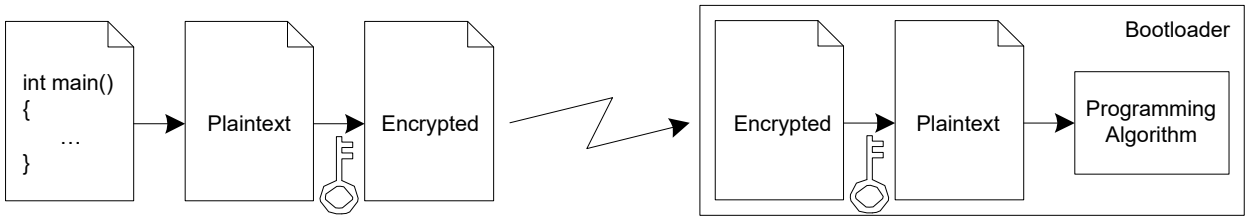
AN2462

AVR231: AES Bootloader

Introduction

This application note describes how firmware can be updated securely on AVR® microcontrollers with bootloader capabilities. The method uses the Advanced Encryption Standard (AES) to encrypt the firmware.

Figure 1. Overview



Features

- Fits AVR Microcontrollers with bootloader capabilities and at least 1 KB SRAM
- Enables secure transfer of firmware and sensitive data to an AVR-based application
- Includes easy-to-use configurable example applications:
  - Encrypting binary files and data
  - Creating target bootloaders
  - Downloading encrypted files to a target
- Implements the Advanced Encryption Standard (AES):
  - 128-, 192-, and 256-bit keys
- AES Bootloader fits into 2 KB
- Typical update times of a 64 KB application, 115200 baud, 3.69 MHz target frequency:
  - AES128: 27 seconds
  - AES192: 30 seconds
  - AES256: 33 seconds
- The application can be evaluated Out Of the Box on ATmega328PB Xplained Mini
- The firmware has been tested to work on the following devices with minimal or no change:
  - ATmega 8/16/162/169/32/64/128/256
  - ATmega168PA
  - ATmega328PB

Table of Contents

1. Description

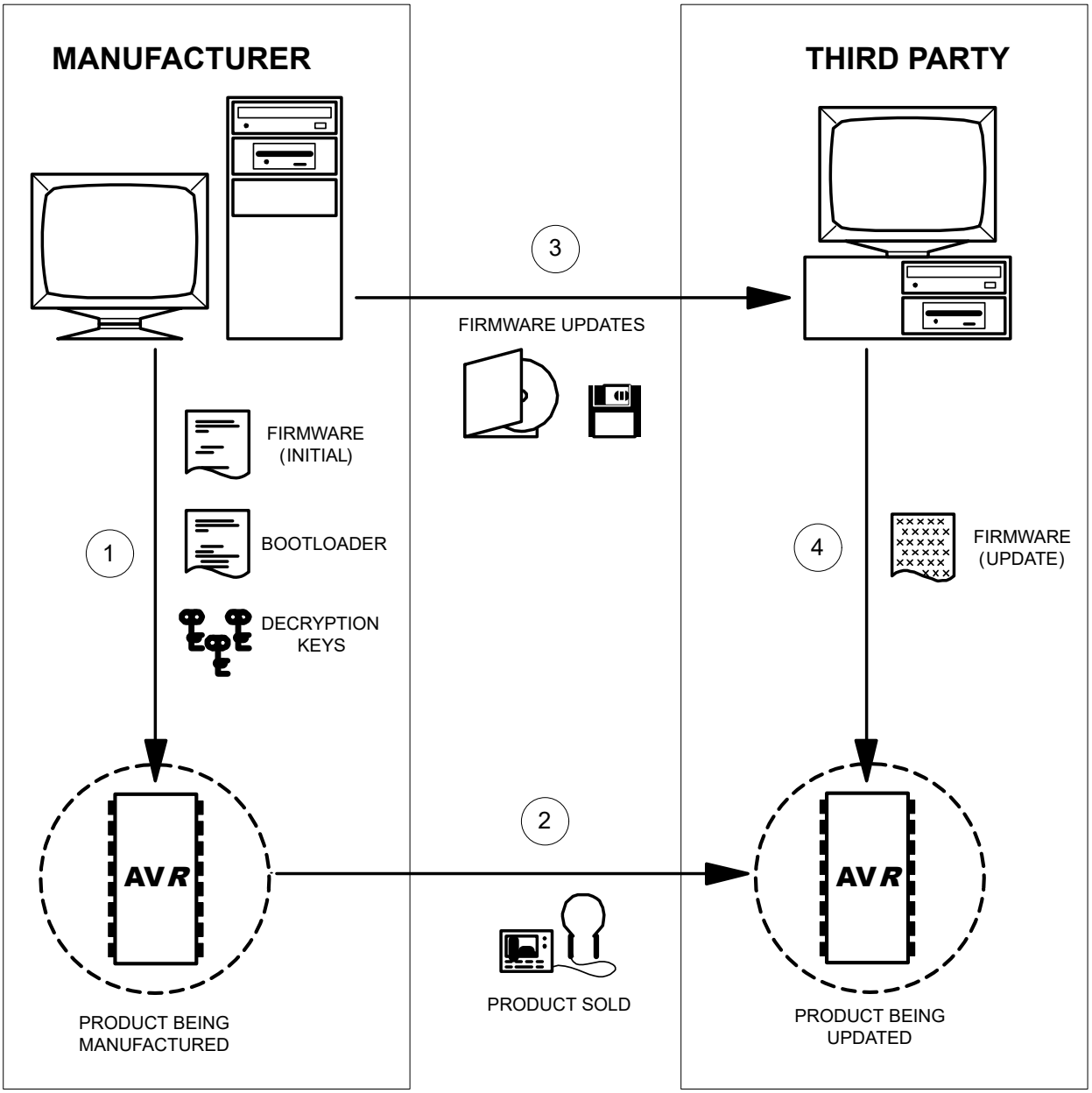
Electronic designs with microcontrollers always need to be equipped with firmware, be it a portable music player, a hairdryer, or a sewing machine. As many electronic designs evolve rapidly there is a growing need for being able to update products that have already been shipped or sold. It may prove difficult to make changes to hardware, especially if the product has already reached the end customer, but the firmware can easily be updated on products based on Flash microcontrollers, such as the AVR. Many AVR microcontrollers are configured such that it is possible to implement a bootloader able to receive firmware updates and to reprogram the Flash memory on demand. The program memory space is divided in two sections; the Bootloader Section (BLS) and the Application Section. Both sections have dedicated lock bits for read and write protection so that the bootloader code can be secured in the BLS while still being able to update the code in the application area. Hence, the update algorithm in the BLS can easily be secured against outside access.

The problem remains with the firmware, which typically is not secure before it has been programmed into Flash memory and lock bits have been set. This means that if the firmware needs to be updated in the field, it will be open for unauthorized access from the moment it leaves the programming bench or manufacturer’s premises.

This application note shows how data to be transferred to Flash and EEPROM memories can be secured at all times by using cryptographic methods. The idea is to encrypt the data before it leaves the programming bench and decrypts it only after it has been downloaded to the target AVR. This procedure does not prevent unauthorized copying of the firmware, but the encrypted information is virtually useless without the proper decryption keys. Decryption keys are stored in only one location outside the programming environment - inside the AVR. The keys cannot be regenerated from the encrypted data. The only way to gain access to the data is by using the proper keys.

The following figure shows an example of how a product is first manufactured, loaded with initial firmware, sold, and later updated with a new revision of the firmware.

Figure 1-1. An Example of the Typical Production and Update Procedure



Note:

1. During manufacturing, the microcontroller is first equipped with a bootloader, decryption keys, and application firmware. The bootloader takes care of receiving the actual application and programming it into Flash memory, while keys are required for decrypting the incoming data. Lock bits are set to secure the firmware inside the AVR.
2. The product is then shipped to a distributor or sold to the end customer. Lock bit settings continue to keep the firmware secured inside the AVR.
3. A new release of the firmware is completed and there is a need to update products, which already have been distributed. The firmware is therefore encrypted and shipped to the distributor. The encrypted firmware is useless without decryption keys and therefore even local copies of the software (for example, on the hard drive of the distributor) do not pose a security hazard.

4. The distributor upgrades all units in stock and those returned by customers (for example, during repairs). The encrypted firmware is downloaded to the AVR and decrypted once inside the microcontroller. Lock bit settings continue to keep the updated firmware secured inside the AVR.

2. Glossary

BLS	Bootloader Section
EEPROM	Electrically Erasable PROM
AES	Advanced Encryption Standard
RAM	Random Access Memory
CBC	Cipher Block Chaining
PC	Personal Computer
CRC	Cyclic Redundancy Check
USART	Universal Synchronous Asynchronous Receiver Transmitter
KB	Kilobytes
LCD	Liquid Crystal Display
IDE	Integrated Development Environment

3. Pre-Requisites

The solutions discussed in this document require basic familiarity with the following:

- Atmel® Studio 7 or later
- ATmega328PB Xplained Mini
- AES concepts
- Bootloader concepts and its implementation in AVR

This application note covers the basic overview of Advanced Encryption Standard (AES). Users who wish to have a better understanding of the concepts of AES are requested to do further reading on relevant literature.

4. Cryptography Overview

The term cryptography is used when information is locked and made unavailable using keys. Unlocking information can only be achieved using the correct keys.

Algorithms based on cryptographic keys are divided into two classes; symmetric and asymmetric. Symmetric algorithms use the same key for encryption and decryption while asymmetric algorithms use different keys. AES is a symmetric key algorithm.

4.1 Encryption

Encryption is the method of encoding a message or data so that its contents are hidden from outsiders. The plain-text message or data in its original form may contain information the author or distributor wants to keep secret, such as the firmware for a microcontroller. For example, when a microcontroller is updated in the field it may prove difficult to secure the firmware against illicit copying attempts and reverse engineering. Encrypting the firmware will render it useless until it is decrypted.

4.2 Decryption

Decryption is the method of retrieving the original message or data and typically cannot be performed without knowing the proper key. Keys can be stored in the bootloader of a microcontroller so that the device can receive encrypted data, decrypt it, and reprogram selected parts of the Flash or EEPROM memory. Decryption keys cannot be retrieved from the encrypted data and cannot be read from AVR microcontrollers if lock bits have been programmed accordingly.

5. AES Overview

5.1 AES Implementation

This section is not intended to be a detailed description of the AES algorithm or its history. The intention is rather to describe the AVR-specific implementations for the various parts of the algorithm. Since memory is a scarce resource in embedded applications, the focus has been on saving code memory. The bootloader application will never be run the same time as the main code, and it is therefore not important to save data memory (RAM) as long as the data memory requirements do not exceed the capacity of the microcontroller.

In the following subsections, some basic mathematical operations and their AVR-specific implementations are described. Note that there are some references to finite field theory from mathematics. Knowledge of finite fields is not required to read this document, but the interested reader should study the AES specification.

Note: If the reader has sufficient knowledge of the implementation of AES, they can skip to 6. Software Implementation and Usage without loss of continuity.

5.1.1 Byte Addition

In the AES algorithm, byte addition is defined as addition of individual bits without carry propagation. This is identical to the standard XOR operation. The XOR operation is its own inverse; hence byte subtraction is identical to addition in the AES algorithm. XOR operations are trivial to implement on AVR.

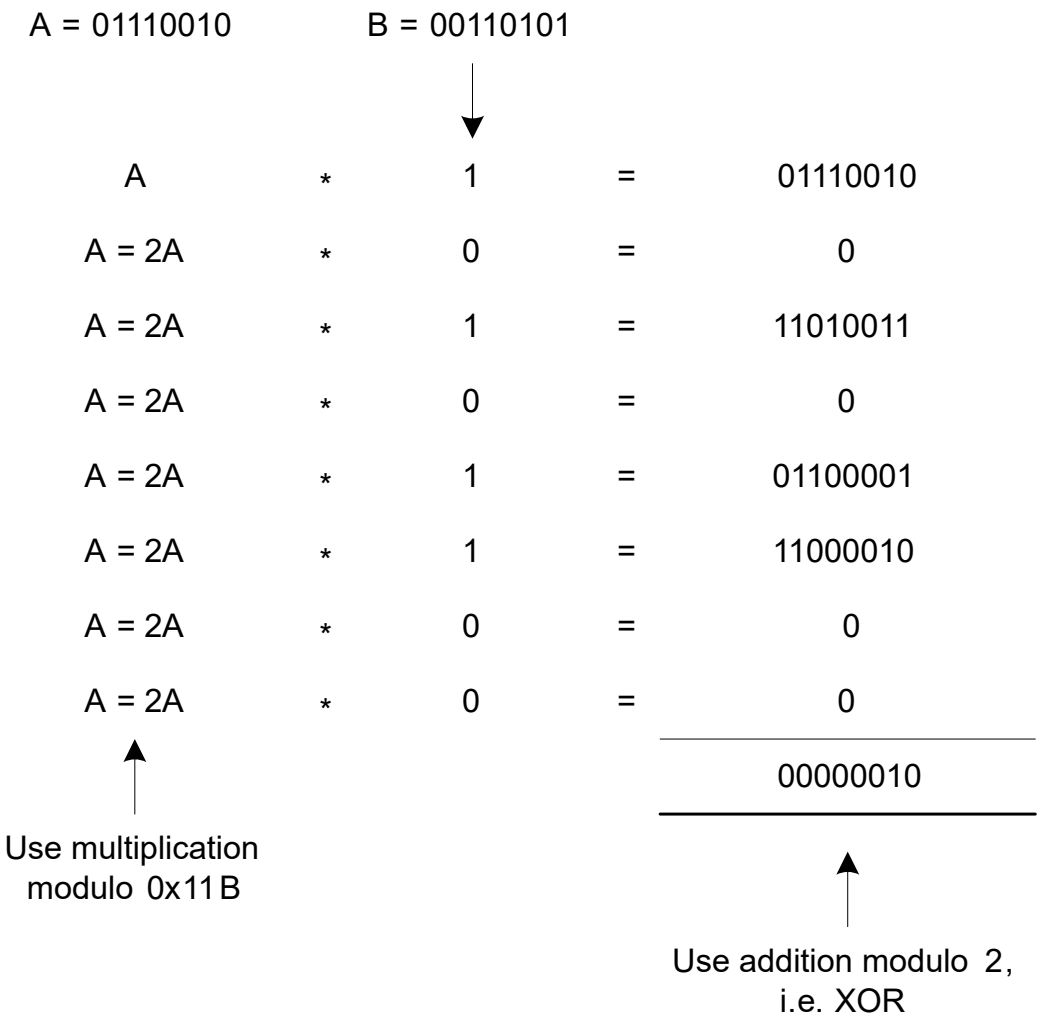
5.1.2 Byte Multiplication

In the AES algorithm, byte multiplication is defined as finite field multiplication with modulus 0x11B (binary 1 0001 1011). A suggested implementation is to repetitively multiply the first factor by 2 (modulo 0x11B) and sum up the intermediate results for each bit in the second factor having value 1.

An example: If the second factor is 0x1A (binary 0001 1010), then the first, third, and fourth intermediate results should be summed.

Another example is shown in the following figure. This method uses little memory and is well suited for an 8-bit microcontroller.

Figure 5-1. Byte Multiplication



The Byte Multiplication can be described by the following pseudo code:

```
bitmask = 1
tempresult = 0
tempfactor = firstfactor
while bitmask < 0x100
    if bitmask AND secondfactor <> 0
        add tempfactor to tempresult using XOR
    end if
    shift bitmask left once
    multiply tempfactor by 2 modulo 0x11B
end while
return tempresult
```

5.1.3 Multiplicative Inverses

To be able to compute finite field multiplicative inverses, that is, 1/x, a trick has been used in this implementation. Using exponentiation and logarithms with a common base, the following identity can be utilized:

Using exponentiation and logarithms to compute 1/x.

$$a^{-\log_a x} = \frac{1}{x}$$

In this case, the base number 3 has been chosen, as it is the simplest primitive root. By using finite field multiplication when computing the exponents and logarithms, the multiplicative inverse is easy to implement. Instead of computing exponents and logarithms every time, two lookup tables are used. Since the multiplicative inverse is only used when preparing the S-box described in 5.1.4 S-Boxes, the memory used for the two lookup tables can be used for other purposes when the S-box has been prepared.

The lookup table computation can be described by the following pseudo code:

```
tempexp = 0
tempnum = 1
do
    exponentiation_table[ tempexp ] = tempnum
    logarithm_table[ tempnum ] = tempexp
    increase tempexp
    multiply tempnum by 3 modulo 0x11B
loop while tempexp < 256
```

5.1.4 S-Boxes

The AES algorithm uses the concept of substitution tables or S-boxes. One of the steps of the algorithm is to apply an invertible transformation to a byte. The S-box is the pre-computed results of this transformation for all possible byte values. The transformation consists of two steps: (1) A multiplicative inverse as described in 5.1.3 Multiplicative Inverses, and (2) a linear transformation according to the following equation, where a~i~ are the bits of the result and b~i~ are the bits of the result from step 1.

Linear transformation used in the S-box:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

A closer look at the matrix reveals that the operation can be implemented as the sum (using XOR addition) of the original byte, the right-hand vector, and the original byte rotated left one, two, three and four times. This method is well suited for an 8-bit microcontroller.

The inverse S-box, used for decryption, has a similar structure and is also implemented using XOR additions and rotations. Refer to the AES specification for the corresponding matrix and to the source code for implementation details.

5.1.5 The ‘State’

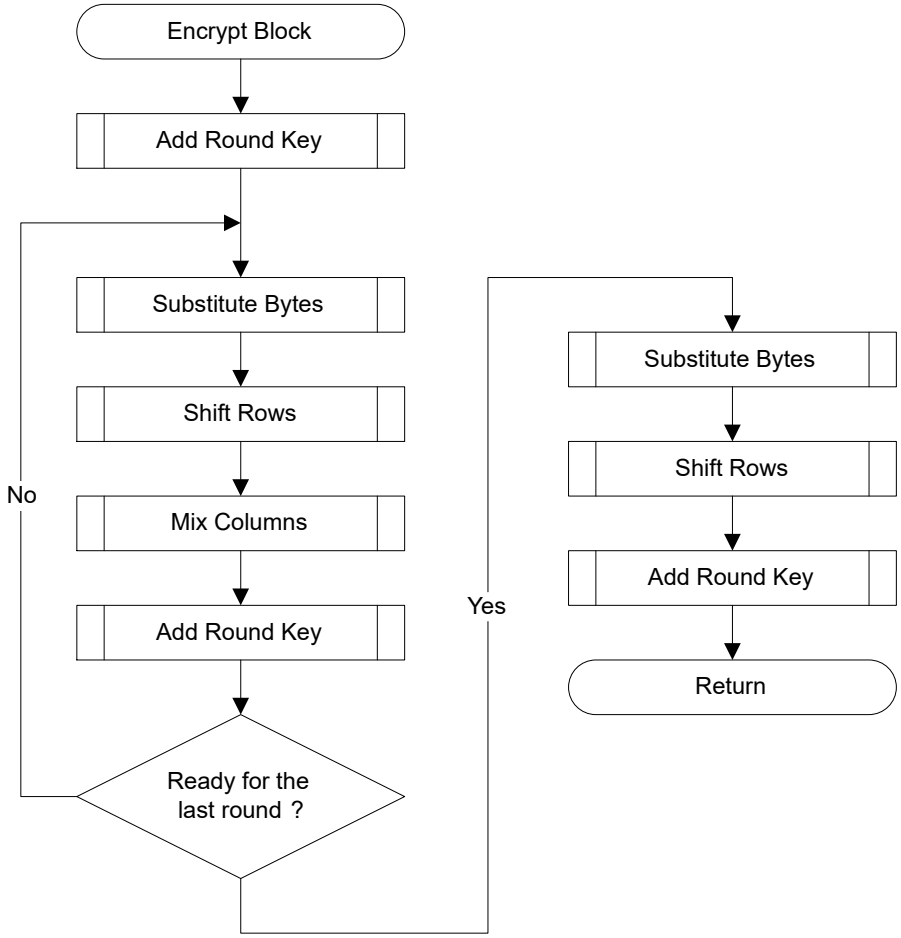
The AES algorithm is a block cipher, which means that data is managed in blocks. For the AES cipher, the block size is 16 bytes. The AES block is often organized in a 4x4 array called the ‘State’ or the ‘State array’. The leftmost column of the State holds the first four bytes of the block, from top to bottom, and so on. The reader should also be aware that in the AES specification, four consecutive bytes are referred to as a word.

5.2 AES Encryption

Before discussing the steps of the encryption process, the concept ‘encryption round’ needs to be introduced. Most block ciphers consist of a few operations that are executed in a loop a number of times. Each loop iteration uses a different encryption key. At least one of the operations in each iteration depends on the key. The loop iterations are referred to as encryption rounds, and the series of keys used for the rounds is called the key schedule. The number of rounds depends on the key size.

The flowchart for the encryption process is shown in the following figure. The following subsections explain the different steps in the process. Each step is implemented as a subroutine for convenience. Using an optimizing compiler removes unnecessary function calls to save code memory.

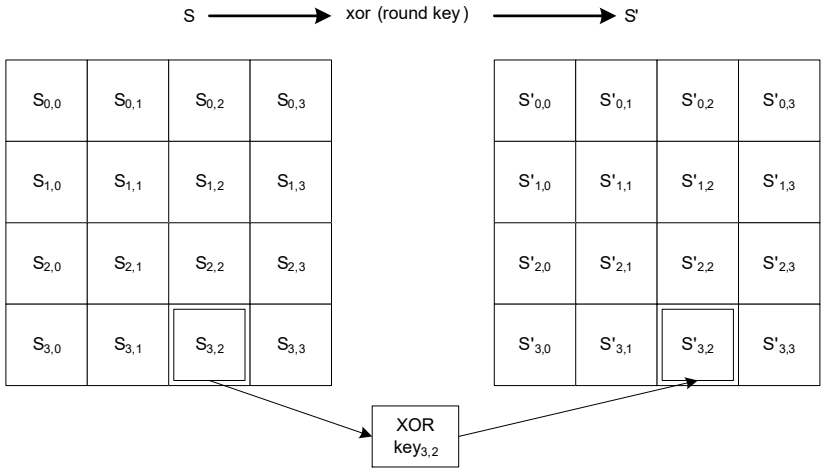
Figure 5-2. Encryption Flowchart



5.2.1 Add Round Key

This step uses XOR addition to add the current round key to the current State array. The round key has the same size as the State, that is, 16 bytes or four words. This operation is implemented as a 16-step loop.

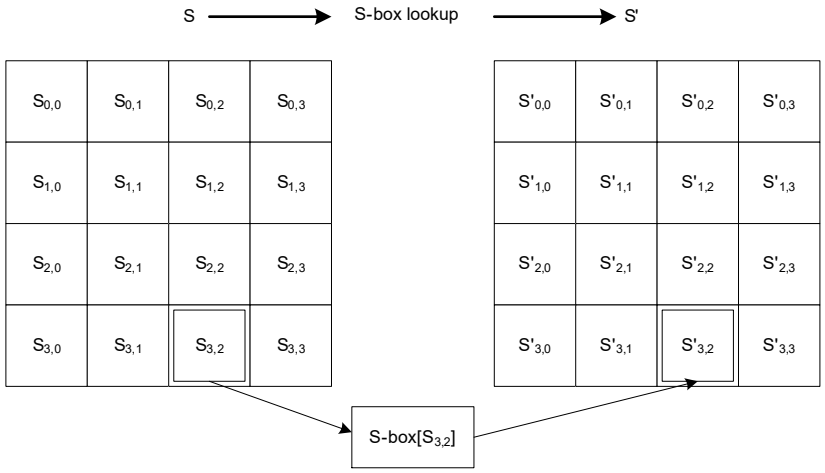
Figure 5-3. Adding the Round Key to the Current State



5.2.2 Substitute Bytes

This step uses the precalculated S-box lookup table to substitute the bytes in the State. This step is also implemented in a 16-step loop like the previous section.

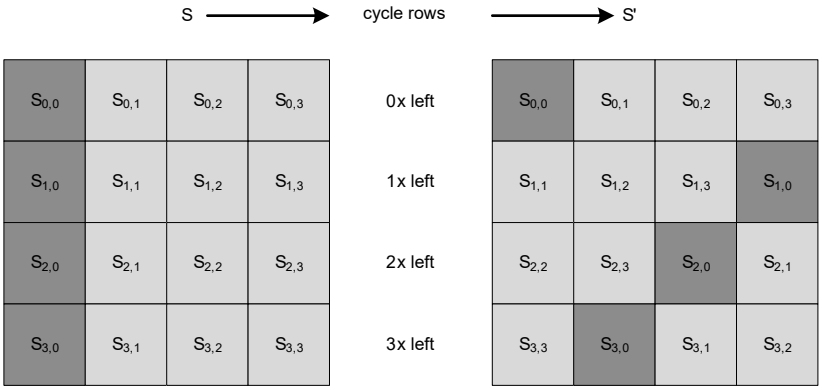
Figure 5-4. Substituting the Bytes of the Current State



5.2.3 Shift Rows

This step operates on the rows of the current State. The first row is left untouched, while the last three are cycled left one, two, and three times, respectively. To cycle left once, the leftmost byte is moved to the rightmost column, and the three remaining bytes are moved one column to the left. The process is shown in the following figure.

Figure 5-5. Cycling the Rows of the Current State

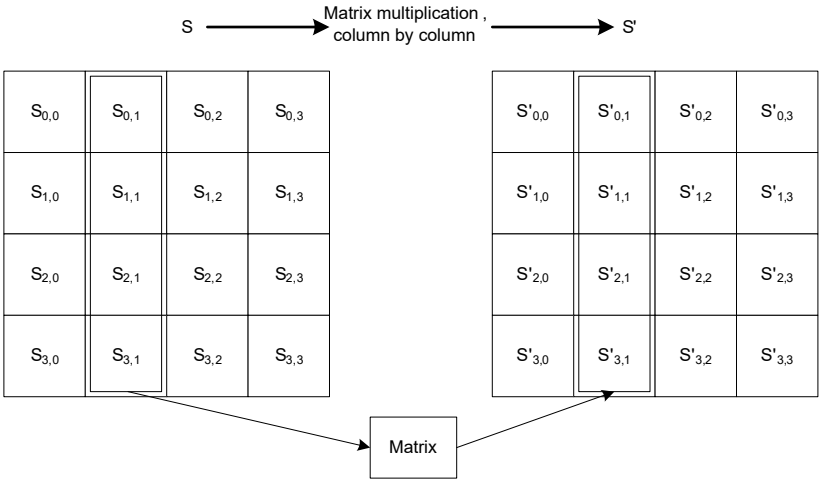


The naive implementation would be to write a subroutine that cycles a row left one time, and then call it the required number of times on each row. However, tests show that implementing the byte shuffling directly, without any loops or subroutines, results in only a small penalty in code size but a significant gain (3x) in speed. Therefore, the direct implementation has been chosen. Refer to the **ShiftRows()** function in the source code for details.

5.2.4 Mix Columns

This step operates on the State column by column. Each column is treated as a vector of bytes and is multiplied by a fixed matrix to get the column for the modified State.

Figure 5-6. Mixing the Columns of the Current State



The operation can be described by the following equation, which are the bytes of the mixed column and are the bytes of the original column.

Figure 5-7. Matrix Multiplication When Mixing One Column

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

This step is implemented directly without any secondary function calls. From the matrix equation, one can see that every byte of the mixed column is a combination of the original bytes and their doubles. Refer to the **MixColumns()** function in the source code for details.

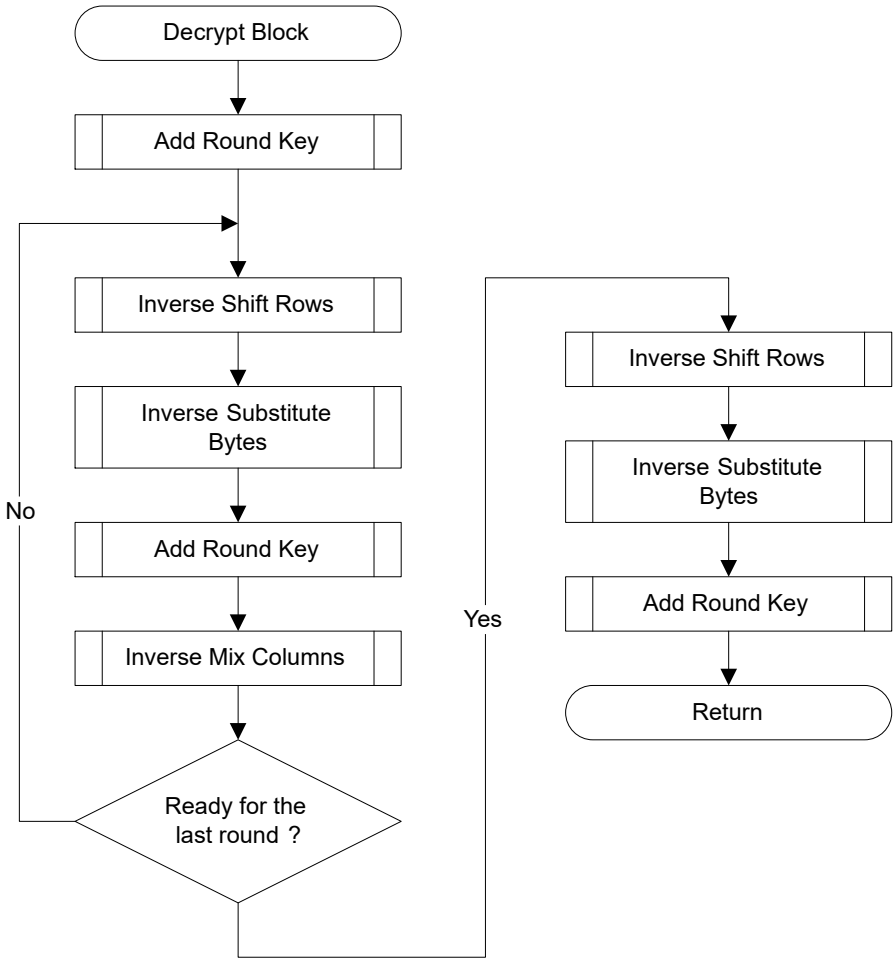
Note: XOR Addition and Finite Field Multiplication from sections 5.1.1 Byte Addition and 5.1.2 Byte Multiplication are used.

5.3 AES Decryption

The process is very similar to the encryption process, except the order of the steps has changed. All steps except “Add Round Key” have their corresponding inverses. “Inverse Shift Rows” cycles the rows right instead of left. “Inverse Substitute Bytes” uses inverse S-boxes.

“Inverse Mix Columns” also uses an inverse transformation. Refer to the AES specification for the corresponding matrix and to the source code for implementation details. The flowchart for the decryption process is shown as follows.

Figure 5-8. Decryption Flowchart

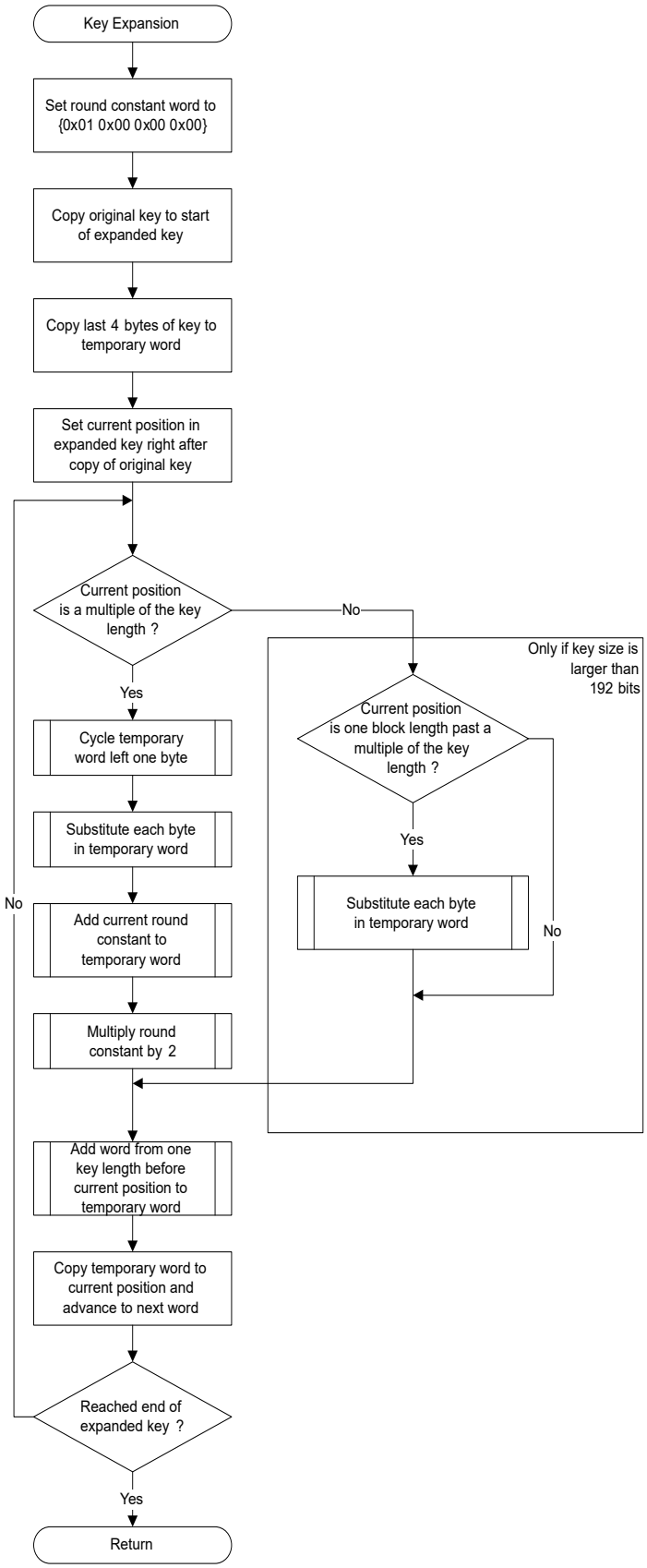


Note: The key schedule used for decryption is the same as for encryption but in reverse order.

5.4 Key Expansion

Key expansion is the process of generating the key schedule from the original 128-, 196-, or 256-bit cipher key. The flowchart for key expansion is shown as follows:

Figure 5-9. Key Expansion Flowchart



The algorithm uses operations already described, such as XOR addition, finite field multiplication, substitution, and word cycling. Refer to the source code for details.

Note: The key expansion is identical for both encryption and decryption. Therefore the S-box used for encryption is required even if only decryption is used. In the AVR implementation, the ordinary S-box is computed prior to key expansion, and then its memory is reused when computing the inverse S-box.

5.5 Cipher Block Chaining – CBC

AES is a block cipher, meaning that the algorithm operates on fixed-size blocks of data. The cipher key is used to encrypt data in blocks of 16 bytes. For a known input block and a constant (although unknown) encryption key, the output block will always be the same. This might provide useful information for somebody wanting to attack the cipher system.

There are some methods commonly used which cause identical plaintext blocks being encrypted to different ciphertext blocks. One such method is called Cipher Block Chaining (CBC).

CBC is a method of connecting the cipher blocks so that leading blocks influence all trailing blocks. This is achieved by first performing an XOR operation on the current plaintext block and the previous ciphertext block. The XOR result is then encrypted instead of the plaintext block. This increases the number of plaintext bits one ciphertext bit depends on.

6. Software Implementation and Usage

This section first discusses some important topics for improving system security. These topics motivate many of the decisions in the later software design.

6.1 Motivation

This application note presents techniques that can be used when securing a design from outside access. Although no design can ever be fully secured it can be constructed such that the effort required to break the security is as high as possible. There is a significant difference between an unsecured design that a person with basic engineering skills can duplicate and a design that only a few, highly skilled intruders can break. In the unsecured case, the design is easily copied and even reverse engineered, violating the intellectual property of the manufacturer and jeopardizing the market potential for the design. In the secured case, the effort required to break the design is so high that most intruders simply focus on developing their own products.

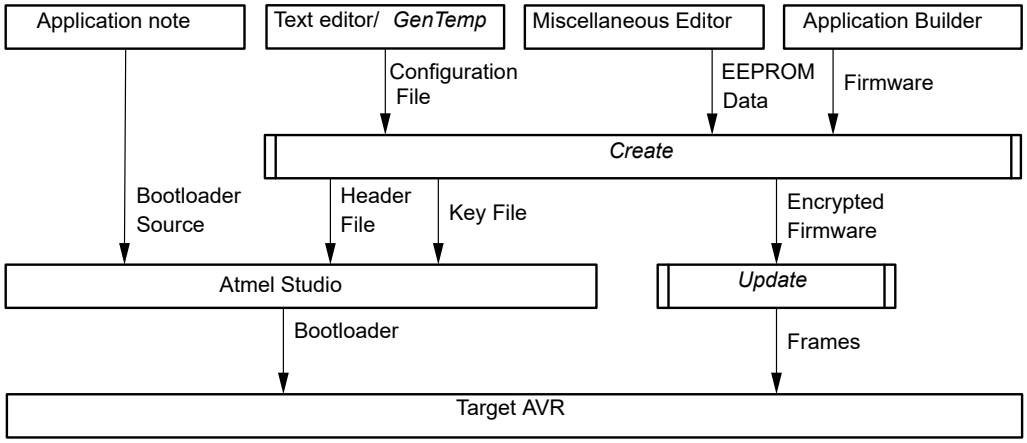
There is only one general rule on how to build a secure system: It should be designed to be as difficult to break as possible. Any mechanism that can be used to circumvent security will be tried during a break attempt. A few examples of what must be considered are given below.

- What will happen if power is removed during a firmware update? What is the state of the microcontroller when power is restored back? Are lock bits and reset vectors set properly at all times?
- Are there any assumptions that can be made on what plain-text data will look like? In order for AES to be broken, there must be a pattern to look for. The attack software will have to be configured to search for a known pattern, such as interrupt vectors at the start of program memory, memory areas padded with zero or one, and so on.
- Is there any feedback that can be derived from the decryption process? Any such feedback can help the attacker. For example, if the decryption algorithm inside the bootloader would give an OK/Not-OK type of signal for each block processed, then this signal could be used as feedback to the attacker.
- Should encrypted frames be sent in another order? If the first frame sent to the bootloader always includes the first block of the encrypted file then the attacker can make some assumptions from this. For example, it can be assumed that the first frame maps program data starting from address zero and that it contains the interrupt vector table. This information helps the attacker to refine the key search. To increase the security of the system, send the frames in random order (the decrypted frames will be mapped to their proper address, anyhow).

6.2 Usage Overview

This and the following subsections describe how to use and configure the applications. The process is illustrated in the following figure.

Figure 6-1. Overview of the Project Flow



The main steps are:

- Create an application for the target AVR. If required, create an EEPROM layout in a separate file.
- Create a configuration file with project dependent information. The application called **GenTemp** can be used for creating a file frame.
- Run the application called **Create**. This will create the header file, key file, and the encrypted file.
- Use Atmel Studio 7 or later, configure and build the bootloader for the target AVR
- Download bootloader to target AVR and set lock and fuse bits
- Now the encrypted firmware may be downloaded to the AVR at any time

6.3 Configuration File

The configuration file contains a list of parameters, which are used to configure the project. The parameters are described in the following table.

Table 6-1. Summary of Configuration File Options

Parameter	Description	Default	Required
PAGE_SIZE	Size of AVR Flash page in decimal bytes. This parameter is part dependent. See the data sheet.	N/A	Yes
KEY1	First part (128-bit) of the encryption key in hex. Should be 16 random bytes, with odd-parity bits inserted after every 8th bit, making a total of 18 bytes.	None: No encryption	No, but strongly recommended
KEY2	Second part (64-bit) of the encryption key in hex. Should be eight random bytes, with odd-parity bits inserted after every 8th bit, making a total of nine bytes. If omitted, AES128 will be used.	None: Use AES128	No, but recommended
KEY3	Third part (64-bit) of the encryption key in hex. Should be nine random bytes, with odd-parity bits inserted after every 8th bit, making a total of	None: Use AES128 or AES192	No, but recommended

Parameter	Description	Default	Required
	nine bytes. If omitted AES128 or AES192 will be used.		
INITIAL_VECTOR	Used for chaining cipher blocks. Should be 16 random bytes in hex.	0	No, but strongly recommended
SIGNATURE	Frame validation data in hex. This can be any four bytes, but it is recommended that the values are chosen at random.	0	No
ENABLE_CRC	Enable CRC checking: YES or NO. If enabled, the whole application section will be overwritten and the application must pass a CRC check before it is allowed to start.	No	No, but recommended
MEM_SIZE	Size of application section in target AVR (in decimal bytes).	N/A	Yes, if CRC is used

The configuration file can be given any valid file name. The name is later given as a parameter to the application that will create the project files. Below is a sample configuration file for the ATmega328PB. The KEY1 parameter is an example 128-bit key (hex 0123456789ABCDEF0123456789ABCDEF) with parity bits inserted.

```
PAGE_SIZE      = 128
MEM_SIZE       = 30720
CRC_ENABLE     = YES
KEY1           = EC38ECC4C11DBC16151A5E346A872AADAD36
INITIAL_VECTOR = F24D994D5DD3E9F1EEE897616C425028
SIGNATURE      = 89DBF334
```

Some of the parameters cannot be set without specific knowledge of the target AVR. The following table summarizes the features of some present AVR microcontrollers with bootloader functionality. For devices not present in this table, refer to the data sheet of the device.

Table 6-2. AVR Feature Summary

Feature	M8	M16	M162	M169	M32	M64	M128	M328PB
Flash Size, bytes	8192	16384	16384	16384	32768	65536	131072	32768
Flash Page size, bytes	64	128	128	128	128	256	256	128
Flash Pages	128	128	128	128	256	256	512	256
BLS (max.), bytes	2048	2048	2048	2048	4096	8192	8192	4096
BLS (max.) Pages	32	16	16	16	32	32	32	32
MEM_SIZE (min.), bytes	6144	14336	14336	14336	28672	57344	122880	28672
PAGE_SIZE, bytes	64	128	128	128	128	256	256	128

6.4 PC Application – GenTemp

This application generates a template for the configuration file. The application generates random encryption keys and initial vectors, leaving other parameters for the user to be filled in (such as the Flash page size). It is recommended to always start with creating a template using this application.

The application is used as follows:

```
gentemp FileName.ext
```

FileName.ext is the name of the configuration file to be created. After the file has been generated it can be edited using any plain text editor.

6.5 PC Application – Create

This application reads information from the configuration file and generates key and header files for the bootloader. It is also used for encrypting the firmware. Typically, the application is run at least twice:

1. To generate key and header files for the bootloader.
2. When new firmware is encrypted.

Note: It is very important that the same encryption information (configuration file) is used when generating project files and encoding the firmware. Otherwise, the bootloader may not have the correct set of encryption keys and cannot decrypt the data. It should also be noted that it is possible to use the information in the configuration file to decrypt the encrypted firmware. Hence, the configuration file must be kept safe at all times and should not be modified after it has been used for the first time.

6.5.1 Command Line Arguments

The following table shows the available command line arguments.

Table 6-3. Summary of Command Line Arguments

Argument	Description
-c <filename.ext>	The path to a configuration file.
-d	If set, contents of each Flash page is deleted before writing. Else, previous data will be preserved if not specifically written to.
-e <filename.ext>	The path to an EEPROM file (data that goes into EEPROM).
-f <filename.ext>	The path to a Flash file (code that goes into Application Section).
-h <filename.ext>	The name of the output header file. This file is later included in the bootloader.
-k <filename.ext>	The name of the output key file. This file is later included in the bootloader.
-l [BLB12] [BLB11] [BLB02] [BLB01]	Lock bits to set. These lock bits are set after all data has been transferred and before control is transferred to the updated application.

Argument	Description
-n	Nonsense. Add a random number of nonsense records to an encrypted file. As nonsense records are ignored by the bootloader, this setting does not affect the application, only the predictability of the output file.
-o <filename.ext>	Output file name. This is the encrypted file that may be distributed and sent to the target when it needs to be updated.

6.5.2 First Run

In the first run, typically, only key and header files for the bootloader are generated. The generation of key and header files is requested using command line arguments. For example:

```
create –c Config.txt –h BootLdr.h –k AESKeys.inc
```

The key and header files must be copied to the project directory of the bootloader application and be included into the bootloader code.

Note: The bootloader project files are preconfigured to use the file names mentioned above, that is, BootLdr.h and AESKeys.inc. It is recommended these file names are not changed.

6.5.3 Subsequent Runs

In subsequent runs, the application is used for encoding the firmware. Prior to encryption, the source file must be compiled, assembled and linked into one code segment file and/or one EEPROM segment file. Files must be of type Intel® hex.

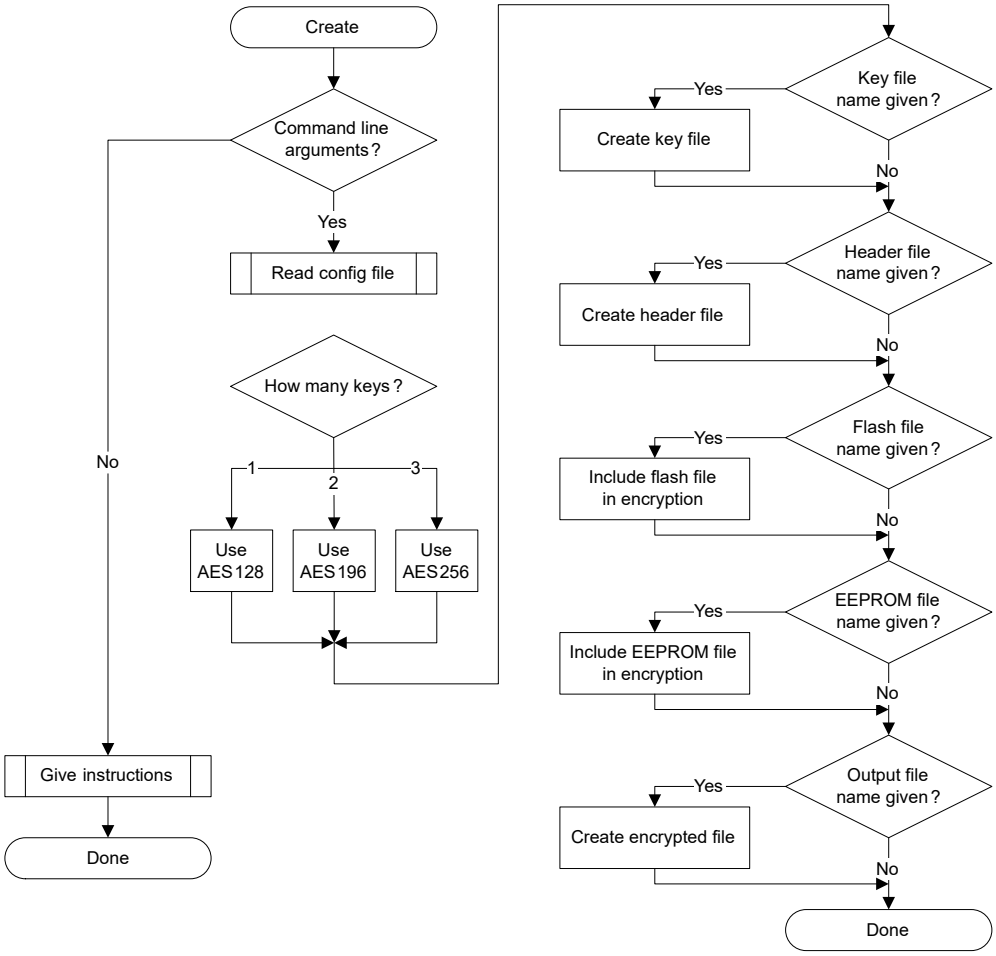
A file name is given at the command prompt and an encrypted file will be generated according to data in the configuration file. For example:

```
create –c Config.txt –e EEPROM.hex –f Flash.hex –o Update.enc –l BLB11 BLB12
```

The application software and EEPROM data files will be combined into a single encrypted file.

6.5.4 Program Flow

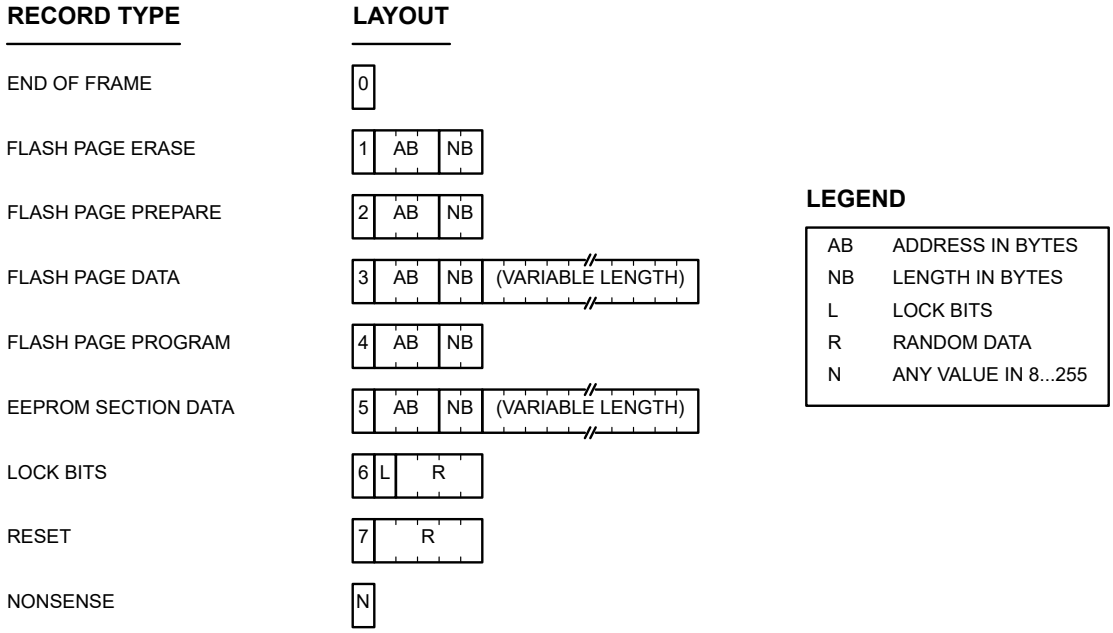
Figure 6-2. Flowchart of the Create Application



6.5.5 The Encrypted File

The Flash and EEPROM files are encrypted and stored in one target file. Before encryption, however, data is organized into records. There are seven types of records, as illustrated in the following figure.

Figure 6-3. Record Types for Encrypted File

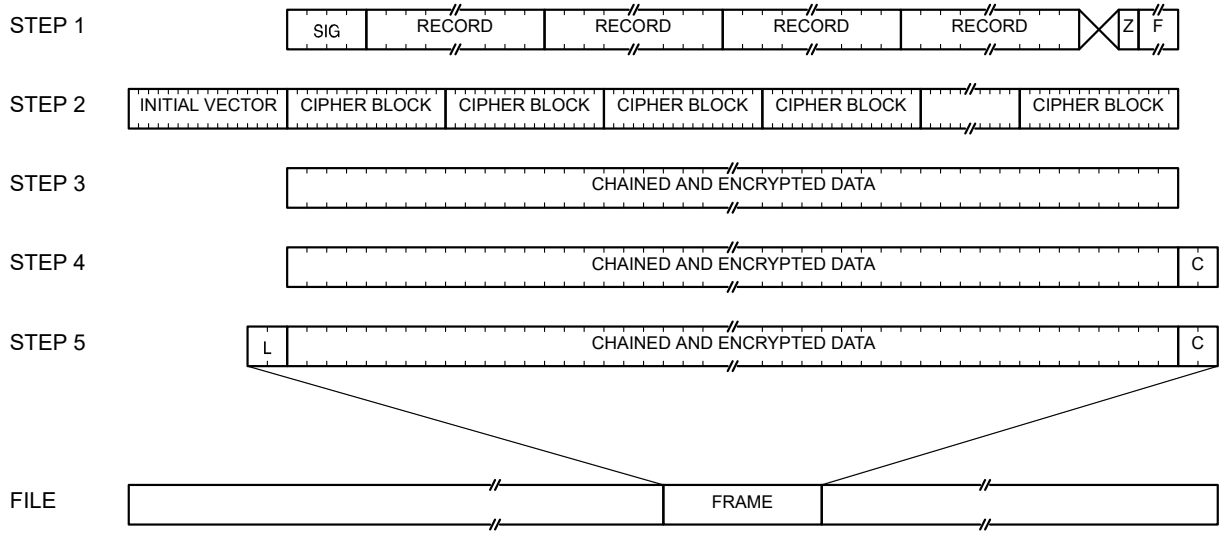


The record type is given as the first byte in the record. The application data is broken down to record types 1, 2, 3, and 4 (that is, erase, prepare, load, and write buffer page to Flash). The data for the EEPROM section is formatted into record type 5. Lock bits are sent in record type 6. Record types 0 and 7 are for ending a frame and transmission, respectively.

All other records, that is, those with a record identifier above 7, are of type nonsense. When this option is enabled (see create tool), a random number of nonsense records will be placed at random locations in the file.

The output file is created as illustrated in the following figure.

Figure 6-4. Creating the Encrypted File



The steps are described below (the numbers refer to the figure above):

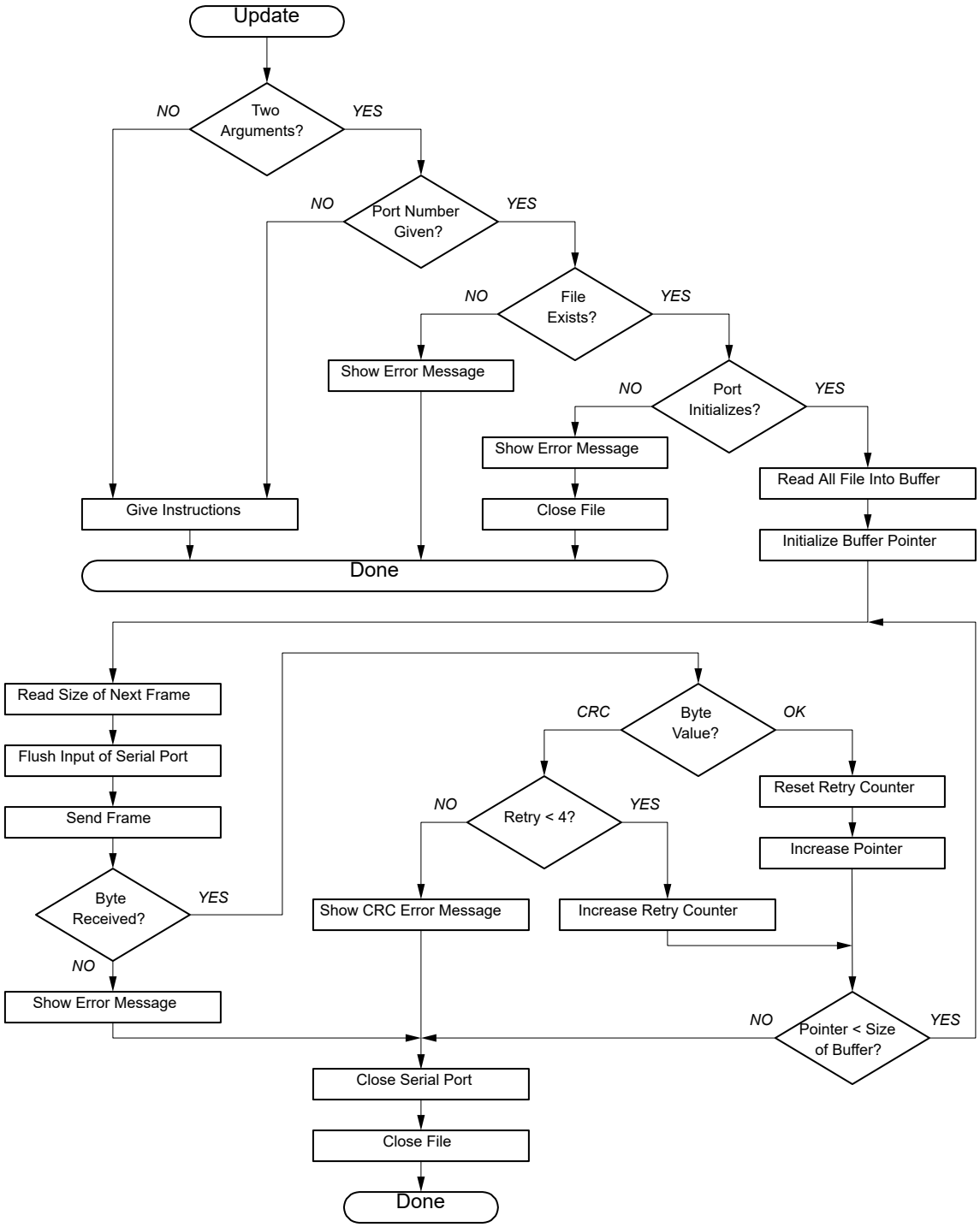
1. Data is formatted into records, which are then lined up following the frame signature (SIG). A zero (Z) is added to mark the end of the frame and the frame is padded with random data (F) to create a frame size that is a multiple of 16 bytes.
2. The initial vector is attached to the frame. In the first frame, the vector is equal to the one given in the configuration file. In subsequent frames, the initial vector is equal to the last cipher block of the previous frame.
3. The initial vector and cipher blocks are chained and encrypted. The initial vector is then removed from the frame.
4. A CRC-16 checksum (C) is calculated and added to the frame.
5. The length (L) of the frame, excluding the length information, is calculated and saved at the start of the frame.

The frame is written to the output file and the procedure is repeated until all data has been processed.

6.6 PC Application – Update

This application is used for sending the encrypted file to the target. The data can be sent via a serial port on the PC directly to the USART on the target hardware. The program flow is illustrated as follows.

Figure 6-5. Overview of the Project Flow



The Update application reads in files generated with the Create application. The file consists of one or more concatenated frames of encrypted data. The application transmits data one frame at a time, pausing in between to wait for a reply from the bootloader. The next frame is transmitted only after an acknowledgment has been received; otherwise, the application will either resend the frame or close communication.

The update application is run from the command prompt. The command prompt arguments are listed in the table below.

Table 6-4. Command Line Arguments for the Update Application

Argument	Description
<filename.ext>	The path to the encrypted file to be transferred
-COM*n*	Serial port, where *n* is the serial port number
-*baudrate*	Baudrate, where *baudrate* is the actual baudrate figure

For example:

```
update blinky.ext -COM1 -115200
```

It should be noted that the update system only updates those parts of the Flash and EEPROM denoted in the application and EEPROM files. If CRC check of the application section is enabled, or the erase option is selected in the create tool, all application memory will be cleared before programming.

7. Hardware Setup

The target hardware must be properly set up before the encrypted firmware can be sent to the bootloader. In this application note, it is assumed that an ATmega328PB Xplained Mini board is used for evaluation. The details on configuration, steps to program, and debug are explained in the following sections. These sections cover only the basic information needed to get an ATmega328PB Xplained Mini up and running. For more information, refer to ATmega328PB Xplained Mini User Guide.

7.1 Connecting the ATmega328PB Xplained Mini Kit

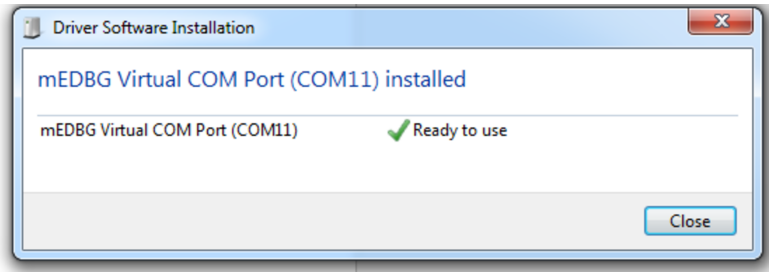
This section helps the user to connect the ATmega328PB Xplained Mini with the Atmel Studio 7.

1. Download and install Atmel Studio version 7 or later versions.
2. Launch the Atmel Studio.
3. Connect the ATmega328PB Xplained Mini to the USB port and it will be visible in the Atmel Studio.

7.1.1 Auto Board Identification of Xplained Mini Kit

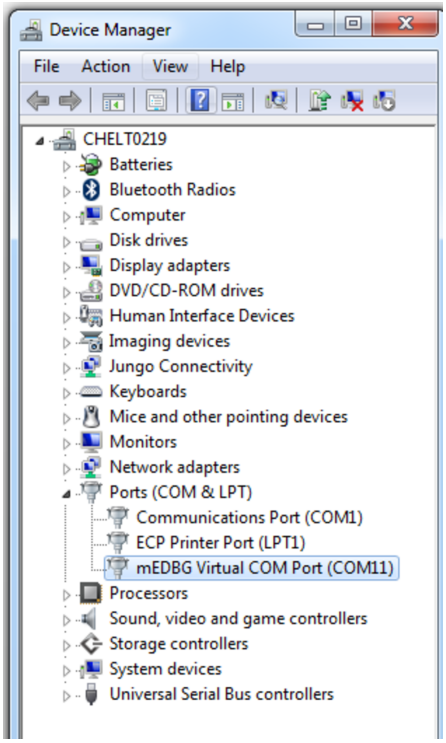
- Once the ATmega328PB Xplained Mini kit is connected to the PC, the Windows® Task bar will pop-up a message as shown below:

  Figure 7-1. ATmega328PB Xplained Mini Driver Installation



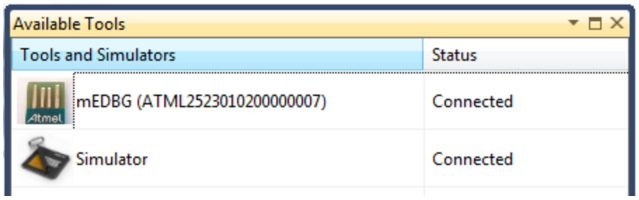
- If the driver installation is proper, EDBG will be listed in the Device Manager as shown below:

Figure 7-2. Successful mEDBG Driver Installation



- Open Atmel Studio and go to **View** → **Available Tools**. The EDBG should get listed in the tools as mEDBG and the tool status should display as **Connected**. This indicates that the tool is communicating properly with Atmel Studio.

Figure 7-3. mEDBG under Available Tools



7.1.2 Connect the ATmega328PB Xplained Mini UART to the mEDBG COM Port

1. Connect the mEDBG USB to the PC.
2. Use the Device Manager to find the COM port number.
3. Default COM port settings are 9600 baud N 8 1. The COM port settings can be changed according to the application by using the Device Manager. This application has been tested with a baud rate of 115200.

7.2 Programming and Debugging

This section helps to program and debug the ATmega328PB Xplained Mini kit by using mEDBG.

7.2.1 Programming the ATmega328PB Xplained Mini By Using mEDBG

1. Connect the mEDBG USB to the PC.
2. Go to the Atmel Studio: Click **Tools**, select **Device Programming**, and then select the connected mEDBG as **Tool with Device** as ATmega328PB and **Interface** as ISP, click **Apply**.

3. Select **Memories** and locate the source .hex or .elf file and then click **Program**.
4. If the source contains fuse settings, go to **Production file** and upload the .elf file and program the fuses.

Note: If ISP programming fails it could be because the debugWIRE is enabled. See 7.2.2 Debugging the ATmega328PB Xplained Mini By Using mEDBG on how to disable debugWIRE mode.

7.2.2 Debugging the ATmega328PB Xplained Mini By Using mEDBG

1. Start **Atmel Studio**.
2. Connect the mEDBG USB to the PC.
3. Open your project.
4. In the **Project** menu, select the project properties page. Select the **Tools** tab, and select mEDBG as debugger and debugWIRE as the interface.
5. In the **Debug** menu click **Start Debugging and Break**.
6. Atmel Studio will display an error message if the DWEN fuse in the ATmega328PB is not enabled, click YES to make Studio set the fuse using the ISP interface.
7. A debug session is started with a break in main. Debugging can start.
8. When exiting debug mode select **Disable debugWIRE and Close** in the **Debug** menu. This will disable the DWEN fuse.

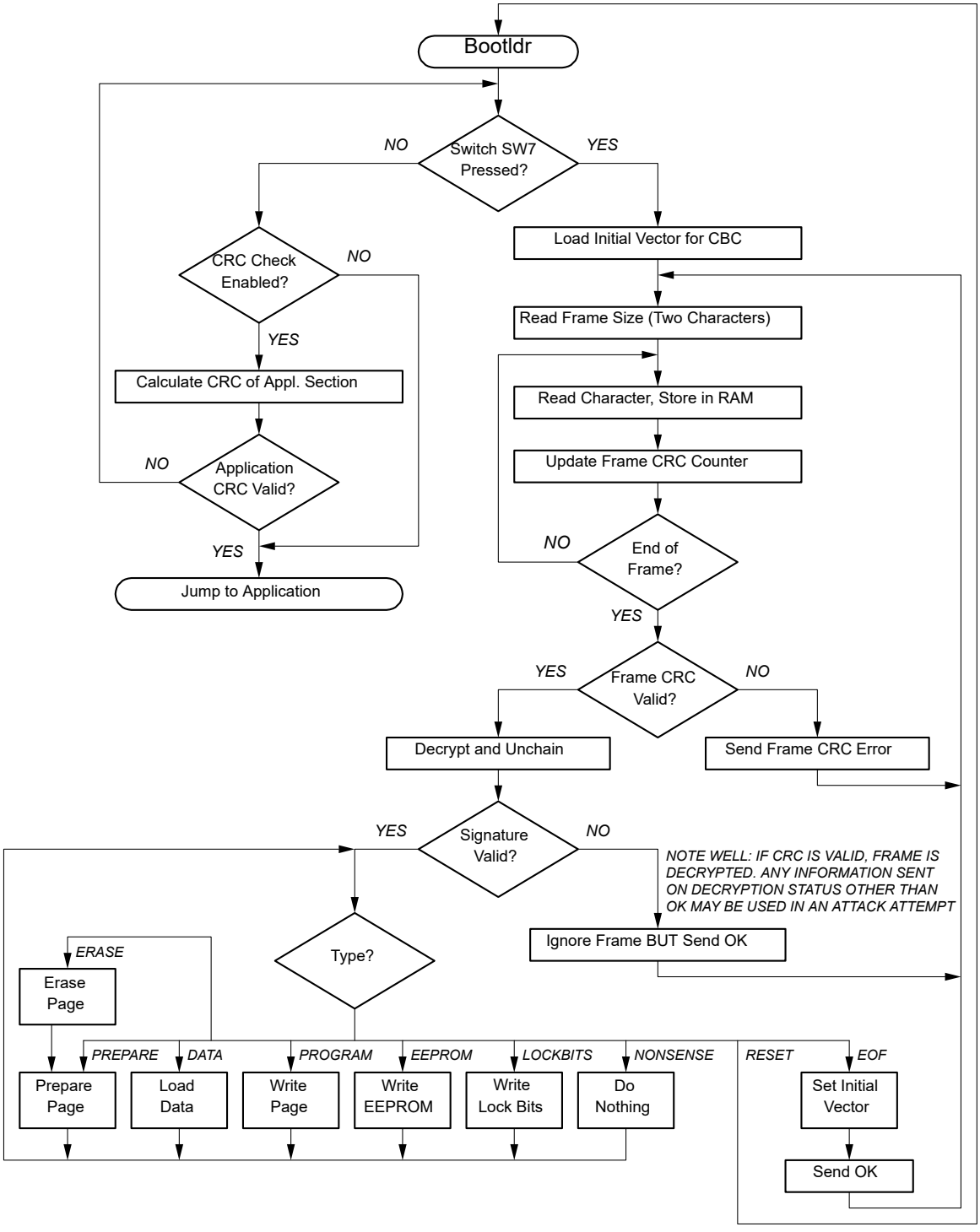
Note:

1. If the debug mode is not exited by selecting **Disable debugWIRE and Close** in the **Debug** menu, the DWEN fuse will be enabled and the target will still be in debug mode, i.e. it will not be possible to program the target by using the SPI (ISP) interface.
2. The bootloader code supplied with this application note has been optimized for size and hence it may not be possible to do debugging. Removing the optimization will increase the code size and it may not fit into the bootloader section. Hence it is recommended to use only the programming mode.

8. AVR Bootloader

The bootloader must reside in the target AVR before the device can be updated with encrypted firmware. The bootloader communicates with the PC and is capable of programming the EEPROM and the application area of the Flash memory. The bootloader included with this application note has been created by using Atmel Studio 7. The program flow is illustrated in the figure below.

Figure 8-1. Flowchart for the AVR Bootloader



8.1 Key and Header Files

Before the bootloader can be compiled, there are some parameters that need to be set up. To start with, the encryption key and target header files generated by the PC application create must be copied to the

bootloader directory. The files will be included when they are referred to with the **#include** directive inside the bootloader source code.

8.2 Project Files

The application note comes with device-specific project files for ATmega328PB.

For other AVR devices, use the project file and modify them as per the steps given in the following section.

8.3 Atmel Studio and IAR Settings

The common settings for Atmel Studio and IAR™ will be listed in this section.

1. The essential fuse settings:
   1.1. The boot reset vector must be enabled.
   1.2. Set Low Fuse Byte to use External Clock @ 16 MHz to make the ATmega328PB example (BAUD RATE 38400) download from Atmel START work properly.
2. The SPMCSR register address has to be defined using SPMREG (refer to the table below) macro in reg.h file, this register address will change if the register is located in the indirect memory region, the address of this register can be found in the register summary section of the data sheet.

Figure 8-2. Setting SPMREG Address in reg.h

```
#ifndef REG_H_
#define REG_H_

// Bit values macros
#define EEMWE 2 // position 2 and value 4   in EEMCR
#define EEWE 1  // position 1 and value 2    in EEMCR

// Bits and their position in SPMREG
#define SPMEN 0 // position zero in SPMCR
#define BLBSET 3
#define PGERS 1
#define PGWRT 2

#ifdef __RAMPZ__
#define _RAMPZ_ 0x3B
#endif

#define SPMREG 0x37
#define SREG 0x3F
#define EEARL 0x21
#define EEARH 0x22
#define EECR 0x1F
#define EEDR 0x20

#endif /* INCFILE1_H_ */
```

3. Other register addresses defined in reg.h file have to be modified according to the device data sheet.

4. Symbol **__RAMPZ__** (refer to the table below) has to be defined in compiler and assembler for the devices with flash size greater than 64 KB.
5. Symbol **__MEMSPM__** (refer to the table below) has to be defined in assembler for the devices whose **SPMCSR** register is located above the direct memory access region (i.e. if SPMCSR register is located above 0x3F). For e.g. ATmega128 has SPMCSR located at 0x68 and hence the definition is required for accessing it.
6. Symbol **BOOT_ADDR=<bootloader section start address>** (refer to the table below) has to be defined in the compiler for the devices with flash size greater than 64 KB.

Table 8-1. Summary of Symbol Settings

Setting	M8	M16	M32	M64	M128	M256	M162	M169	M328PB	M168PA
__RAMPZ__	No	No	No	No	Yes	Yes	No	No	No	No
__MEMSPM__	No	No	No	No	Yes	No	No	No	No	No
BOOT_ADDR	No	No	No	No	0x1E000	0x3E000	No	No	No	No
BOOT_START(IAR)	0x1800	0x3800	0x7000	0xE000	0x1E000	0x3E000	0x3800	0x3800	0x7000	0x3800
.text(Atmel Studio)	0x1800	0x3800	0x7000	0xE000	0x1E000	0x3E000	0x3800	0x3800	0x7000	0x3800
SPMREG	0x37	0x37	0x37	0x37	0x68	0x37	0x37	0x37	0x37	0x37

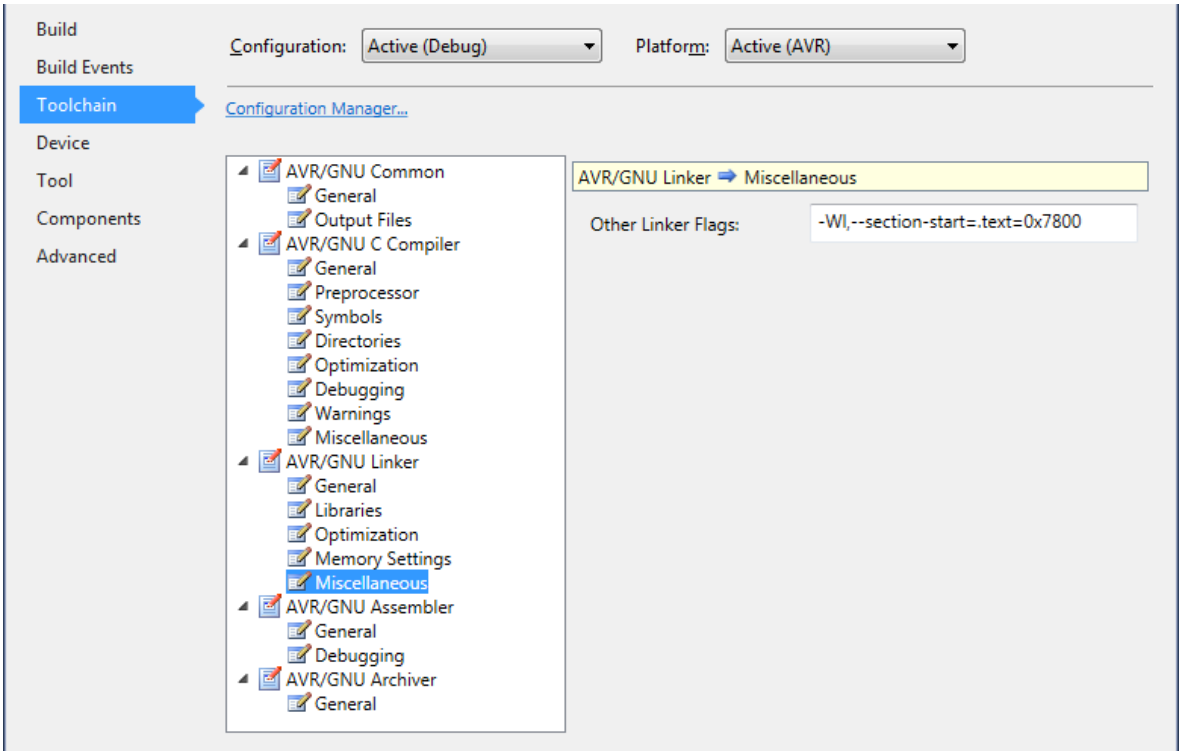
Note: The value of BOOT_ADDR,BOOT_START and .text in the table above is the value when the boot size is maximum. You can change this value according to the actual boot size.

8.3.1 Atmel Studio Settings

The compiler, linker, and assembler setting, shown below are required while configuring for a specific device in Atmel Studio.

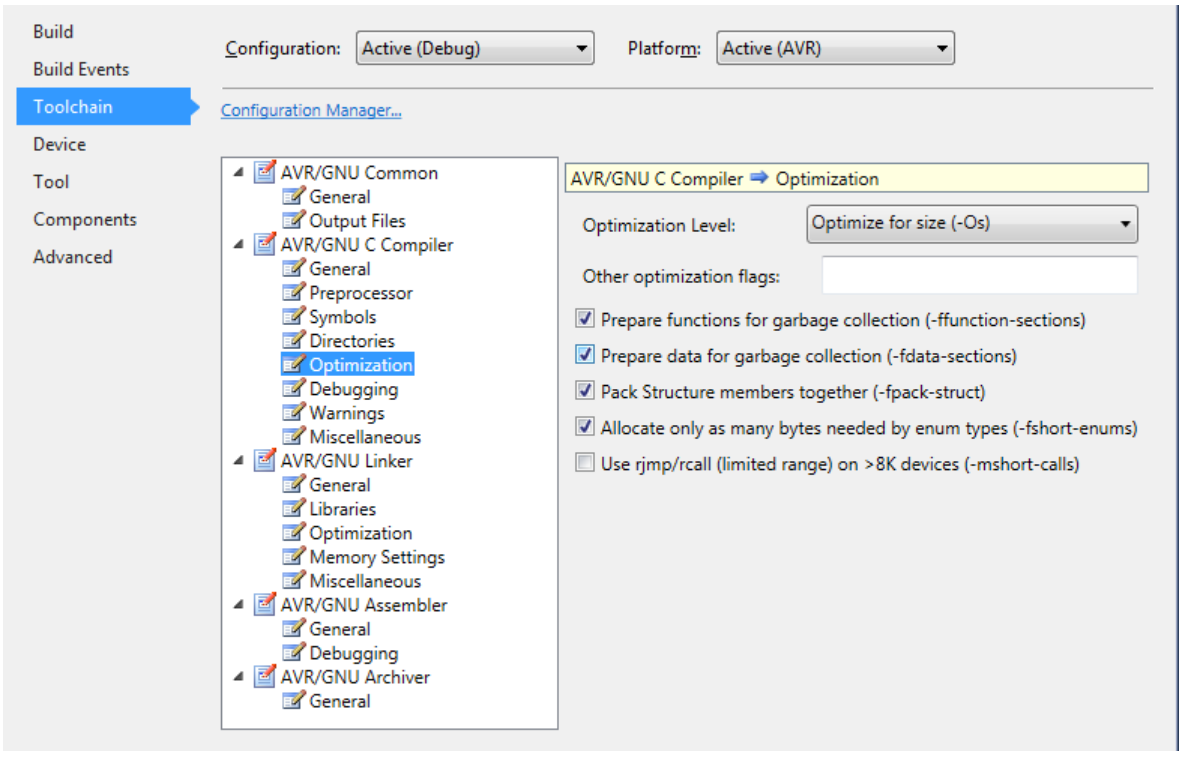
1. Bootloader start address .text (refer to Table 8-1) has to be defined in the linker setting for linking the code to the bootloader start address (as shown in the figure below). Start address should be given as byte address (multiplying word address by 2). For e.g., if bootloader word address of ATmega328PB is 0x3C00, corresponding byte would be 0x3C00 * 2 = 0x7800.

Figure 8-3. Setting Bootloader Address Code Link Address



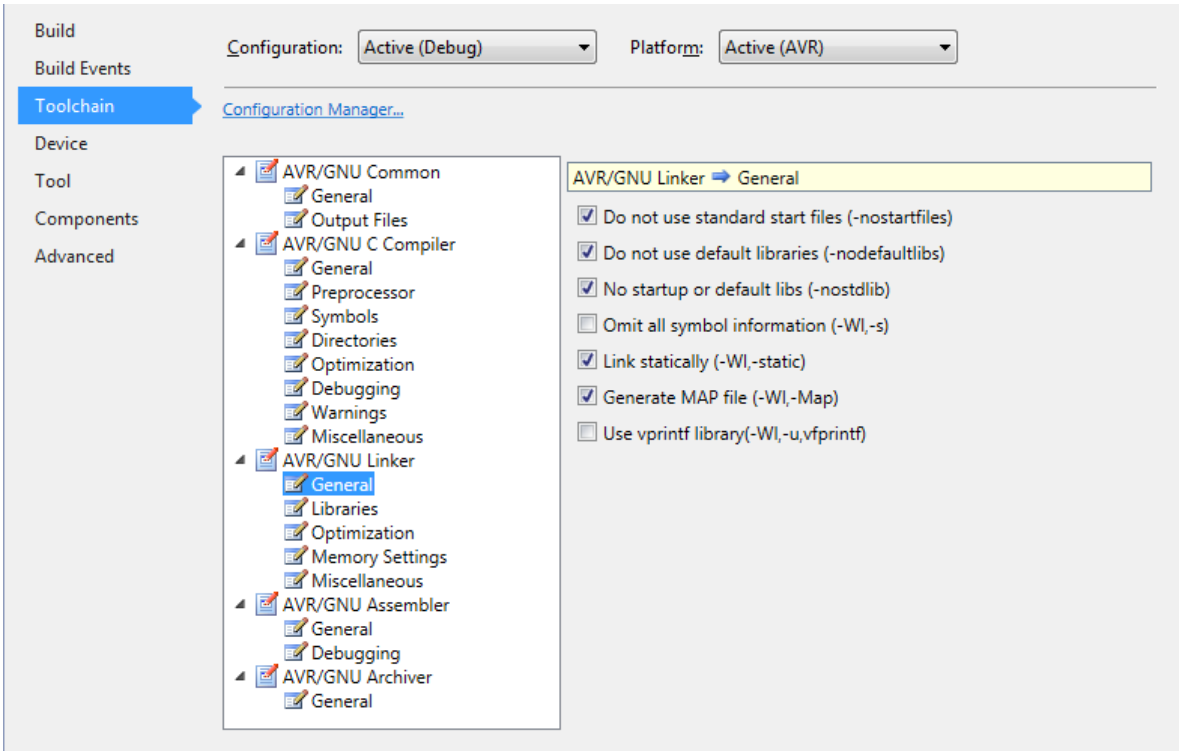
2. Optimization Settings (as shown in the figure below).

Figure 8-4. Optimization Settings



3. Other Linker Settings (as shown in the figure below).

Figure 8-5. Other Linker Settings

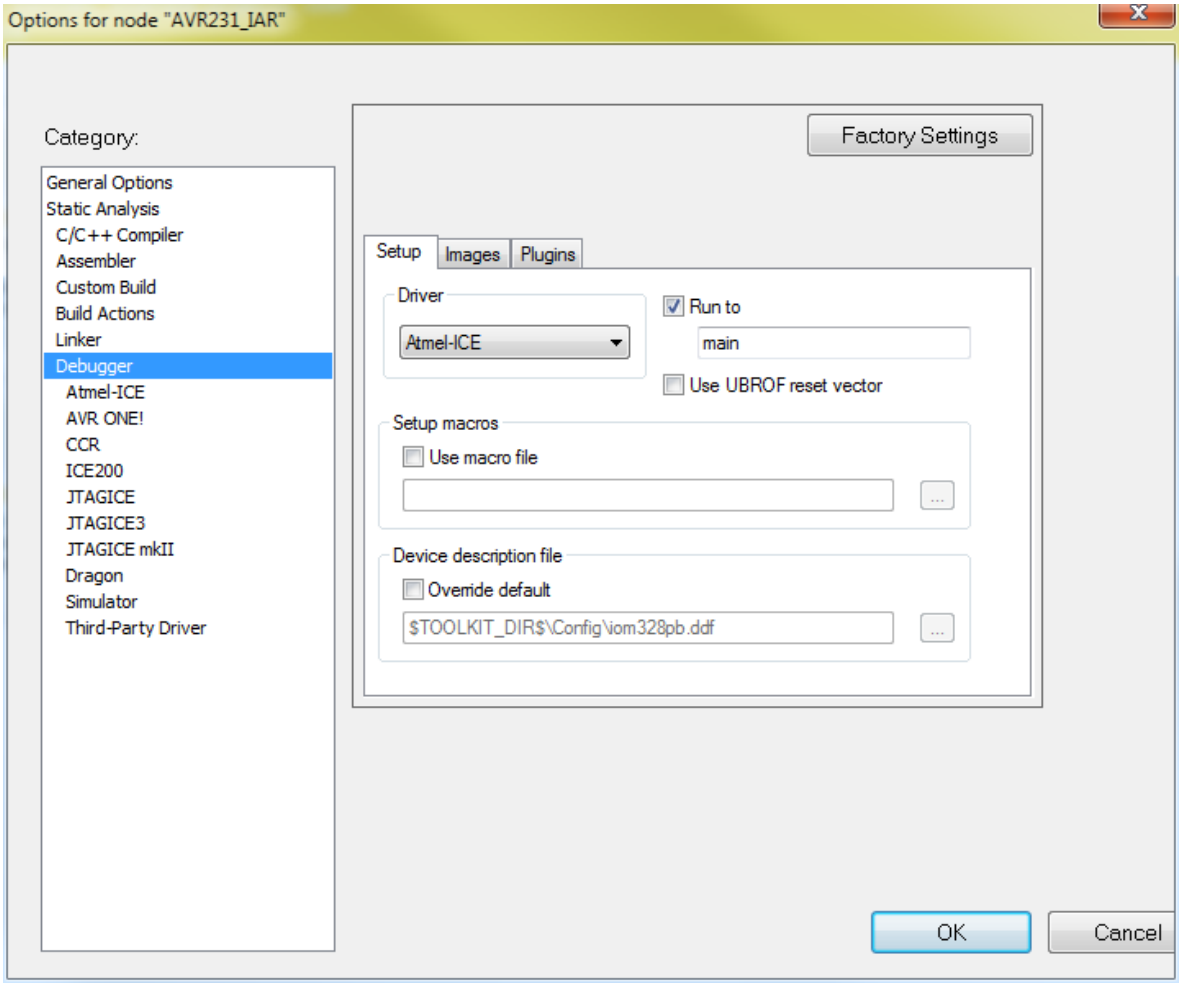


8.3.2 IAR Settings

The compiler, linker, and assembler settings shown below are required while configuring for a specific device in IAR.

1. When using the mEDBG in the ATmega328PB Xplained Mini kit, the Atmel-ICE should be selected in Debugger Driver setting (as shown in the figure below).

Figure 8-6. Debugger Driver



2. A user-defined .xcl file should be used to override default linker file. Follow the steps below to make the linker work properly:

   2.1. Copy the template linker file (for example, C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.4\avr\src\template\cfgm328pb.xcl) according to the device to your project and rename it what you like.

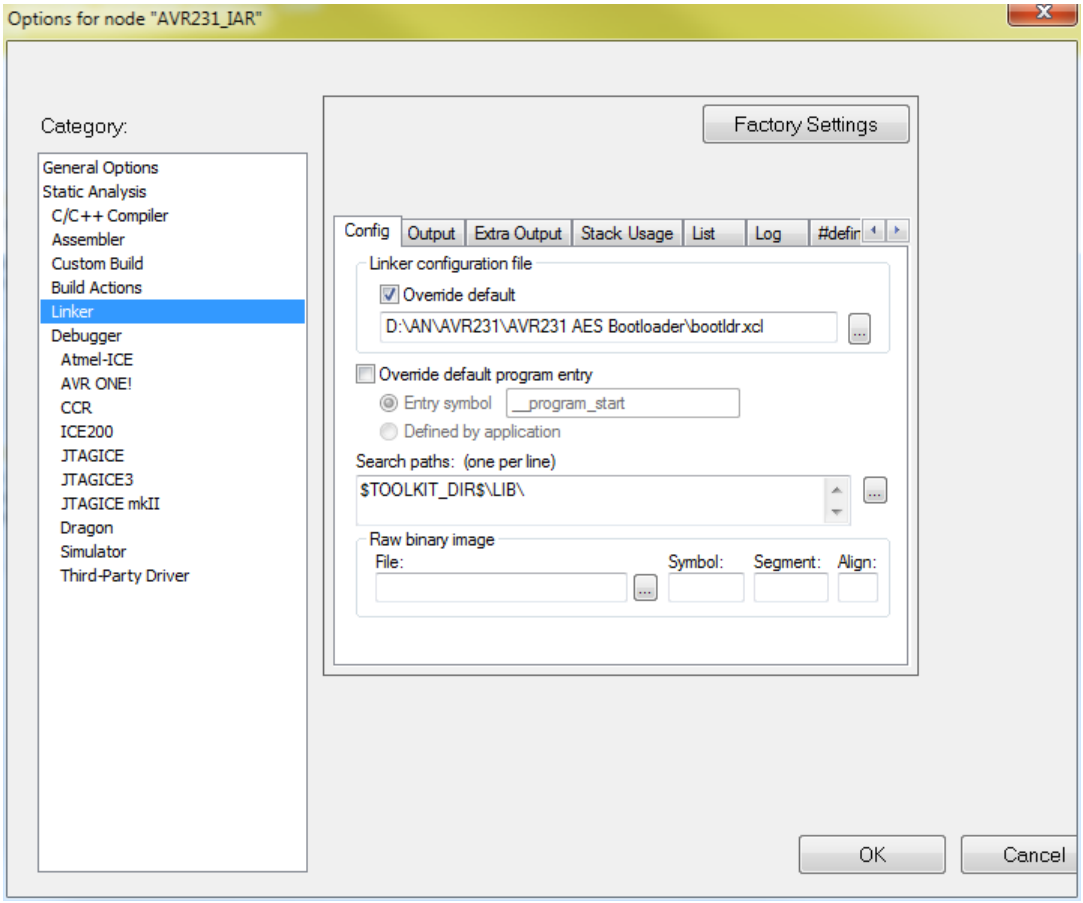
   2.2. Open this file and paste the below content under the original content and save it.

```
-ca90
-w29


//==========================================================================
// Interrupt vectors

-Z(CODE)INTVEC=BOOT_START-(BOOT_START+_..X_INTVEC_SIZE-1)
//-H1895
-h(CODE)BOOT_START-(BOOT_START+_..X_INTVEC_SIZE-1)

//==========================================================================
// Code memory

-Z(CODE)NEAR_F,HUGE_F,SWITCH,INITTAB,DIFUNCT,CODE=BOOT_START-_..X_FLASH_END
-Z(FARCODE)FAR_F,FARCODE=BOOT_START-_..X_FLASH_END


//==========================================================================
// RAM
```

```
-Z(DATA)NEAR_I,NEAR_Z=_..X_SRAM_BASE-_..X_SRAM_END
-Z(DATA)RSTACK+_..X_RSTACK_SIZE=(_..X_SRAM_END-_..X_RSTACK_SIZE+1)-_..X_SRAM_END
-Z(DATA)CSTACK+_..X_CSTACK_SIZE=(_..X_SRAM_END-_..X_RSTACK_SIZE-_..X_CSTACK_SIZE
+1)-(_..X_SRAM_END-_..X_RSTACK_SIZE)
//-Z(DATA)RSTACK+40=(_..X_SRAM_END-40+1)-_..X_SRAM_END
//-Z(DATA)CSTACK+300=(_..X_SRAM_END-40-300+1)-(_..X_SRAM_END-40)
//-Z(DATA)TINY_I,TINY_Z,TINY_N=RAM_BASE-FF
//-Z(DATA)TINY_I,TINY_Z,TINY_N=RAM_BASE-100
```

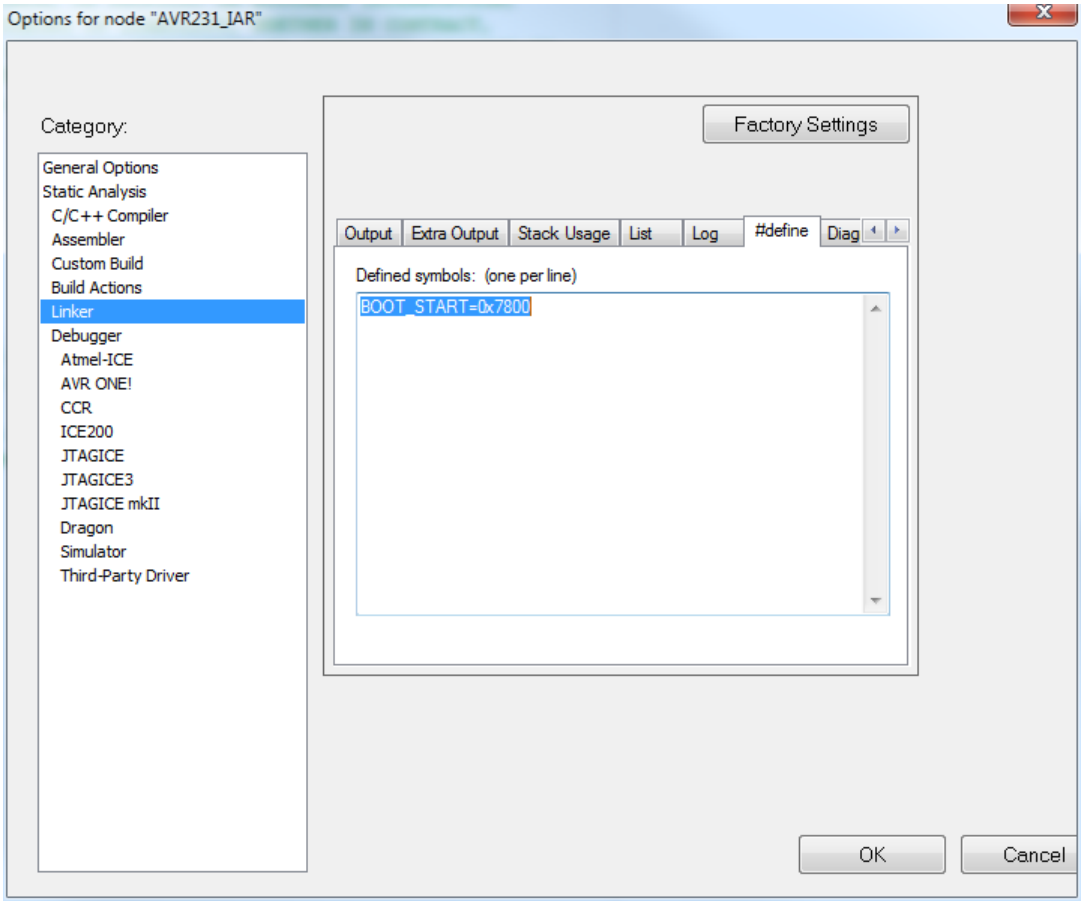
2.3. Use this file to override the default linker file (as shown in the figure below).

Figure 8-7. Override Default Linker File



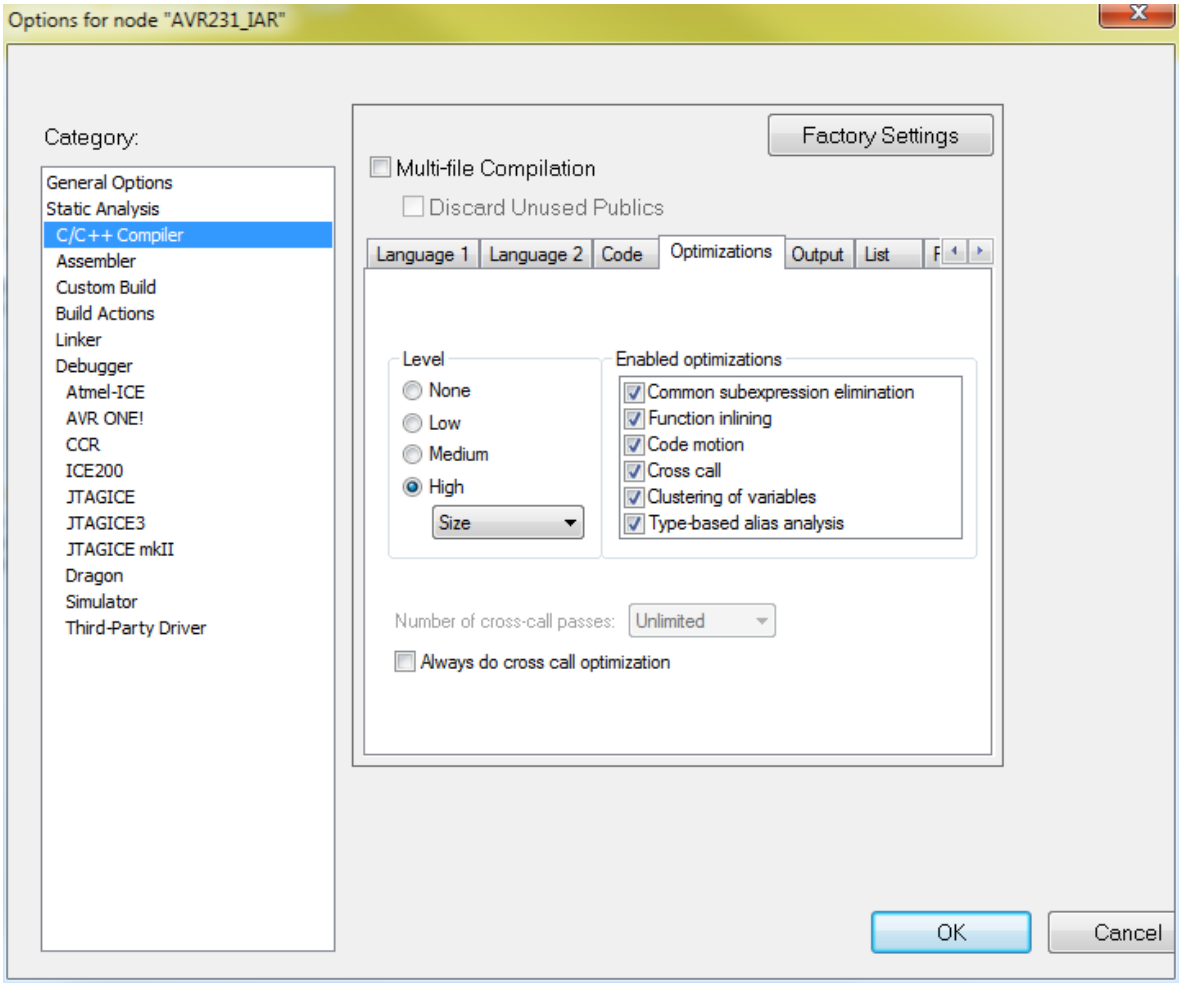
2.4. Define BOOT_START (refer to Table 8-1) according to the device (as shown in the figure below).

Figure 8-8. Linker #define



3. Optimization Settings (as shown in the figure below).

Figure 8-9. Optimization Settings



8.4 Installing the Bootloader

Compile the bootloader and then download it to the target using Atmel Studio 7 or later. Before running the bootloader, the following fuse bits must be configured:

- Size of Boot Loader Section. Set fuse bits so that the section size matches the BOOT_SIZE setting, as described earlier. Note that the BLS is usually given in words, but the BOOT_SIZE parameter is given in bytes.
- Boot reset vector. The boot reset vector must be enabled.
- Oscillator options. The oscillator fuse bits are device dependent. They may require configuration (affects USART).

Note: Pay special attention in setting oscillator options correctly. Even a small misadjustment could result in communication failure.

Recommended fuse bit settings are provided in the table below. See the device data sheet for detailed explanation of device dependent fuse bits.

Table 8-2. Recommended Fuse Bits

	M8, M8515, M8535, M16, M162, M169, M32, M64	M128	328PB
BOOTSZ1:0	0:0	0:1	0:1
BOOTRST	0	0	0

Note: “0” means programmed, “1” means not programmed.

It is recommended to program lock bits to protect both the application memory and the bootloader, but only after fuse bits have been set. Lock bits can be programmed using Microchip IDE (Atmel Studio 7 or later). BLS lock bits will also be set during firmware update, provided that they have been defined as command line arguments when the firmware is encrypted. The recommended lock bit settings are:

- Memory lock bits: These should be set to prevent unauthorized access to memory. Note that after the memory has been locked it cannot be accessed via in-system programming without erasing the device.
- Protection mode for Boot Loader Section: SPM and LPM should not be allowed to write to or read from the BLS. This will prevent the firmware in the application section to corrupt the bootloader and will keep the decryption keys safe.
- Protection mode for application section: No restrictions should be set for SPM or LPM accessing the application section; otherwise the bootloader cannot program it.

Note: It is important to understand that if the device is not properly locked then the memory can be accessed via an ISP interface and the whole point of encrypting the firmware is gone.

Recommended lock bit setting for present AVR MCUs are given in the table below. See the device data sheet for a detailed explanation of lock bits.

Table 8-3. Recommended Lock Bits

	M8, M8515, M8535, M16, M162, M169, M32, M64, M128	328PB
BLB12 : BLB11	0 0	0 0
BLB02 : BLB01	1 1	1 1
LB2 : LB1	0 0	0 0

8.5 Performance

The following sections summarize the system performance with respect to execution time and code size.

8.5.1 Execution Time

The time required for the target device to receive, decode, and program data depends on the following factors:

- File size. The more data, the longer it takes.
- Baudrate. The higher the transmission speed, the shorter the transmission time.
- Target AVR speed. The higher the clock frequency, the shorter the decoding time.

- Programming time of Flash page. This is a device constant and cannot be altered.
- Key-size. AES128 is faster to decrypt than AES256. In fact, AES192 is slower than AES256. It has something to do with 192 not being a power of 2.
- Miscellaneous settings. For example, CRC check of application section takes short time.

8.5.2 Code Size

Using the highest optimization setting for the compiler, the bootloader will fit nicely into 2 KB of Flash memory.

It should be noted that if no encryption keys are given, the bootloader is built without AES support. This application note then performs as a standard bootloader system and can be used on any AVR with bootloader support.

9. Summary

This application note has presented a method for transferring data securely to an AVR microcontroller with bootloader capabilities. This document has also highlighted techniques that should be implemented when building a secured system. The following issues should be considered in order to increase the security of an AVR design.

Implement a bootloader that supports downloading in encrypted form. When the bootloader is first installed (during manufacturing) it must be equipped with decryption keys, required for future firmware updates. The firmware can then be distributed in an encrypted form, securing the contents from outsiders.

Use AVR lock bits to secure Application and Boot Loader sections. When lock bits are set to prevent reading from the device, the memory contents cannot be retrieved. If lock bits are not set, there is no use encrypting the firmware.

Encrypt the firmware before distribution. Encrypted software is worthless to any outside the entity without the proper decryption keys.

Keep encryption keys safe. Encryption keys should be stored in two places only: in the bootloader, which has been secured by lock bits, and in the firmware development bench at the manufacturer.

Chain encrypt data. When data is chained, each encrypted block depends on the previous block. As a consequence, equal plaintext blocks produce different encrypted outputs.

Avoid standard, predictable patterns in the firmware. Most programs have a common framework and any predictable patterns, such as an interrupt vector table starting with a jump to a low address, only serve to help the intruder. Also, avoid padding unused memory areas with a constant number.

Hide the method. There is no need to mention which algorithm is being used or what the key length is. The less the intruder knows about the system, the better. It may be argued that knowing the encryption method fends off some attackers, but knowing nothing about the method increases the effort and may fend off even more.

The bootloader may also be used to erase the application section, if required. Many attack attempts include removing the device from its normal working environment and powering it up in a hacking bench. Detecting, for example, that an LCD is missing or that there are CRC errors in the memory, the bootloader may initiate a complete erase of all memory (including the bootloader section and decryption keys).

In applications where it is not feasible or possible to use an external communications channel for updates, the firmware can be stored in one of the CryptoMemory® devices. The memory can be packaged as a removable smart card, which can easily be inserted in a slot of the device when an upgrade is needed. The microcontroller can check for the presence of a CryptoMemory upon start-up and retrieve a firmware upgrade as needed.

Use secure hardware. A strong encryption protocol is useless if the hardware has structural flaws. There are no reported security issues with the AVR microcontrollers.

This list can be made much longer but the purpose of it is merely to set the designer off in the right direction. Do not underestimate the wit or endurance of your opponent.

10. Get Source Code from Atmel | START

The example code is available through Atmel | START, which is a web-based tool that enables configuration of application code through a Graphical User Interface (GUI). The code can be downloaded for both Atmel Studio and IAR Embedded Workbench® via the direct example code-link(s) below or the *BROWSE EXAMPLES* button on the Atmel | START front page.

Atmel | START web page: http://start.atmel.com/

Example Code

- AVR231 AES Bootloader:
  - http://start.atmel.com/#example/Atmel:AVR231_AES_Bootloader:0.0.1::Application:AVR231_AES_Bootloader:

Press *User guide* in Atmel | START for details and information about example projects. The *User guide* button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel | START project configurator.

Atmel Studio

Download the code as an .atzip file for Atmel Studio from the example browser in Atmel | START, by clicking *DOWNLOAD SELECTED EXAMPLE*. To download the file from within Atmel | START, click *EXPORT PROJECT* followed by *DOWNLOAD PACK*.

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel | START user guide, select *Using Atmel Start Output in External Tools*, and *IAR Embedded Workbench*. A link to the Atmel | START user guide can be found by clicking *About* from the Atmel | START front page or *Help And Support* within the project configurator, both located in the upper right corner of the page.

11. References

- ATmega328PB data sheet (http://www.microchip.com/wwwproducts/en/atmega328pb)
- ATmega328PB Xplained Mini kit (http://www.microchip.com/developmenttools/productdetails.aspx?partno=atmega328pb-xmini)
- Atmel Studio (http://www.atmel.com/tools/atmelstudio.aspx?tab=overview)
- Atmel START (http://start.atmel.com)
- AT10764 (http://ww1.microchip.com/downloads/en/appnotes/atmel-42508-software-library-for-aes-128-encryption-and-decryption_applicationnote_at10764.pdf)
- AVR230 DES Bootloader (http://ww1.microchip.com/downloads/en/appnotes/doc2541.pdf)
- Handbook of Applied Cryptography (http://cacr.uwaterloo.ca/hac)
- AES Specification (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf)

12. Revision History

Doc Rev.	Date	Comments
B	04/2018	Section "Get Source Code from Atmel START" is added, with the link to Atmel START example code.
A	06/2017	Converted to Microchip format and replaced the Atmel document number 2589. The document is updated and the application code is tested for the ATmega328PB device.
2589E	03/2012	New template. Section "PC Application" is updated. Fixed wrong interrupt vector size in the ATmega32 project. Removed cycle dependance on data.
2589D	08/2006	Some minor fixes.

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-2831-2

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820