AN2570

Secure UART Bootloader for SAM D10

Introduction

Many modern embedded systems require firmware updates to fix errors or to support new features. Simultaneously, protection of the intellectual property plays an important role. Microcontrollers have robust firmware protection mechanisms; however, the firmware is vulnerable to interception during the data transfer from the external source.

One way to solve this problem is to use a secure Bootloader and distribute only encrypted images of the firmware to the public. This application note describes the design and operation of a secure Bootloader for SAM D10 devices, as well as describing the encryption algorithm.

Features of a Secure Bootloader

- Secure
- Small size (2 Kbytes)
- Uses UART RX and TX pins, and an optional Bootloader Entry pin
- Allows self updating
- Firmware integrity verification

Table of Contents

1. Hardware Configuration

The UART pins used by the Bootloader depend on the device type, which are listed in the following table.

Table 1-1. Hardware Configuration

Device	UART TX	UART RX	Entry
SAM D10	PA10	PA11	PA25

The Bootloader Entry pin has an active low-level. The value of the pin is sampled at the beginning of the Bootloader execution. Although the internal pull-up resistor is enabled before sampling the pin, it is recommended that the Bootloader Entry pin be pulled up externally for improved noise immunity.

The UART setting used by the secure Bootloader is 115200 8,N,1. This Bootloader supports UART baud rate auto-tuning feature, which may be useful for hosts that cannot set an exact value of the baud rate. Auto-tuning can be initiated at any point of operation. To perform auto-tuning, the host must send a break signal (at least 11 bits of low-level) followed by a character 0x55. No response is expected for this request.

2. Method of Entry

The Bootloader can be invoked in a number of ways:

1. The Bootloader will run automatically if there is no valid firmware. The firmware is considered valid if the first word is not 0xFFFFFFFF. Normally, this word contains an initial stack pointer value, hence the first word will never be 0xFFFFFFFF unless the device is erased.
2. The Bootloader will run on external request if the value of the Bootloader Entry pin is low on the Bootloader execution start.
3. The Bootloader will run on an application (internal) request if the first four locations in the SRAM are equal to 0x78656C41.

The external reset takes priority over any other method of entry.

The following code can be used by the application to request the Bootloader execution:

```
{
  uint32_t *ram = (uint32_t *)HSRAM_ADDR;
  __disable_irq();
  ram[0] = 0x78656c41;
  ram[1] = 0x78656c41;
  ram[2] = 0x78656c41;
  ram[3] = 0x78656c41;
  NVIC_SystemReset();
}
```

3. Bootloader Commands

All Bootloader commands have the same general format, as shown in the following table.

Table 3-1. Bootloader Commands

Command ID	Guard Value	Data 0	...	Data N
1 byte	4 bytes	4 bytes	...	4 bytes

The number and meaning of the data words varies with the command. All data words must be sent in a little-endian (LSB first) format.

The Guard Value must be a constant value of 0x78656C41, which provides additional protection against spurious commands.

All bytes of the command frame must be sent within 100 ms of each other. After 100 ms of idle time, an incomplete command is discarded and the Bootloader goes back to waiting for a new Command ID. This behavior allows the host to resynchronize if synchronization loss.

The Bootloader understands the following commands:

- `Unlock (0xA0)`
- `Data (0xA1)`
- `Verify (0xA2)`
- `Reset (0xA3)`

The `Unlock` command must be issued before the first `Data` command. and it has the following payload:

- Data 0 – Starting Offset
- Data 1 – Image Size
- Data 2 – Data 5 – Nonce

The Starting Offset is the offset from the beginning of the Flash memory. To upgrade the Bootloader itself this value must be set to zero. The application image offset is device-dependent and valid values are listed in the following table. The image offset must be aligned at an Erase Unit Size boundary, which is also device-dependent. The image size must be in increments of Erase Unit bytes.

Table 3-2. Valid Values for Application Image Offset

Device	Application Offset, bytes	User area Offset, bytes	Erase Unit Size, bytes
SAM D10	2048	1792	256

The Nonce is an arbitrary 16-byte value that must never repeat for a given key. The number may be an incrementing counter, or a sequence of random bytes, or a combination of both. When random numbers are used, ensure that they are derived from a good source of entropy. The details on using this number for encryption and authentication are outlined in Appendix A.

The Data command is used to send image data and it has the following payload:

- Data 0 – Starting Offset
- Data 1 – Data N – Image Data (Erase Unit Size bytes)
- Data N+1 – Data N+5 – Image MAC

A Starting Offset must be located inside the region previously unlocked by the `Unlock` command. Any attempts to request the write outside of the unlocked region will result in an error, and the supplied data will be discarded.

Image data must be encrypted and authenticated. Image Message Authentication Code (MAC) ensures that data is authenticated and is not damaged during transfer. MAC also ensures that data belongs to the current location and the current file. Details on the generation, use of this number for authentication, and encryption are outlined in Appendix A.

The secure Bootloader supports simultaneous Flash memory write and reception of the next block of data. The next block of data may be transmitted as soon as the status code is returned for the first one.

Due to this behavior, the status code for the last block will be sent before this block is written into the Flash memory. To ensure that this block is written, the host must send another command and wait for the response. Therefore, the `Verify` or `Reset` command must be sent after the last block of data.

The user area is located in the last block of the Bootloader. The master encryption key is stored in the first 16 bytes of the user area, and the remainder may be used to store arbitrary user data (serial numbers, additional security information, and so on). The user area is updated similar to the regular blocks of data, hence it requires full security processing. This means that the old key must be used for encryption and authentication of the new key. If the old key is lost, then it is impossible to recover the device using this Bootloader.

The default encryption key is `00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f`.

The `Verify` command is used to verify the image data, and the `Verify` command has no payload. The image integrity during the transfer is ensured through the MAC, and the Bootloader reads back the data after it is written into the Flash memory. The `Verify` command may be issued at any time during the image upload, it will return the current value of the verification status.

The `Reset` command is used to exit the Bootloader and run the application, which is necessary if the host has no control over the reset pin. However, it can also be useful even if the host has control over the reset, because this command allows the host to communicate up to four words of arbitrary information to the application. It has the following payload:

- Data 0 – Arbitrary Value 0
- Data 1 – Arbitrary Value 1
- Data 2 – Arbitrary Value 2
- Data 3 – Arbitrary Value 3

The supplied arbitrary values are passed to the application in the first four locations in the SRAM.

All the supplied values must not be equal to 0x78656C41, as this will request the Bootloader execution.

4. Response Codes

The Bootloader will send a single character response code in response to each command. Sequential commands can only be sent after the response code is received for a previous command, or after a 100 ms time-out without a response.

The following are the possible response codes:

- OK (0x50) – Command received and processed successfully
- Error (0x51) – There were errors during the processing of the command
- Invalid (0x52) – Invalid command is received
- Verification OK (0x53) – Image verification is successful
- Verification Fail (0x54) – Image verification is failed.

5. Programming Algorithm

After issuing each command, the host must wait for the response code for at least 100 ms. If no response code is received during this time, the command may be considered lost and may be repeated again.

The host controller must perform the following actions to update the firmware:

1. Request the Bootloader entry.
2. Wait for at least 5ms for the Bootloader to start.
3. Issue the `Unlock` command with the required image parameters.
4. Send the `Data` command with the firmware data.
5. Repeat step four until the entire image is transferred.
6. Issue the `Verify` command and check the response code.
7. If the response code is verification fail, repeat the update starting from step 3.
8. Issue the `Reset` command.

6. Appendix A: Security Procedures

The Bootloader is using the Spritz stream cipher. Spritz is a replacement for a well known RC4 cipher, but it has a number of important improvements over RC4. In addition to fixing some biases and vulnerabilities in RC4, Spritz can be used to implement a cryptographic hash function, a deterministic random number generator, and an encryption algorithm with authentication of the encrypted data. Spritz is friendly for low-end embedded systems, because it mostly relies on basic arithmetic operation over 8-bit values and table lookups.

Spritz is called a “spongy cipher" as it is based on a sponge function. In general, sponge algorithms take an input bit stream of any size, and produce an output of any desired size. New data may be “absorbed” at any point during the generation of the output stream. This property, applied in various ways, allows for construction of the basic security primitives.

In practice, Spritz inputs and outputs are limited to 8-bit chunks to align with byte streams. Therefore, Spritz takes an arbitrary input byte stream, and produces an output byte stream based on the internal state.

All possible functions of the Spritz cipher are built from a small set of basic functions that modify internal state in various ways. The most useful basic building blocks are:

- `InitializeState()` – Resets the state of the algorithm to a well known state
- `Absorb(M)` – Absorbs the message M (array of bytes), and shuffles the internal state
- `AbsorbStop()` – Absorbs a special STOP symbol, that is not a part of the input alphabet
- `Squeeze(n)` – Returns a requested number of pseudo-random bytes

`AbsorbStop()` is a very distinctive feature of the Spritz algorithm. This function modifies internal state in a way that would not be possible by absorbing any character from the regular alphabet.

A common part of many security algorithms is calculation of the hash value of the concatenation of multiple byte streams. Those byte streams may represent an encryption key and a message. For example, if concatenated streams can have variable length, there is a possibility of hash collision from two valid input combinations. For instance, the value of *Hash(FearNot* || *Secured)* is the same as the value of *Hash(Fear* || *NotSecured)*.

To solve this problem, algorithms typically require padding of the password to a known fixed length. In case of Spritz, `AbsorbStop()` can be used as a concatenation symbol, so this collision becomes impossible, even without padding.

In the case of the Bootloader, all inputs have a predefined size, so `AbsorbStop()` is not used, since it would be redundant.

Security in a Bootloader is a very different application from a typical use for the ciphers. A lot of attacks are based on observation of large volumes of encrypted data to successfully exploit a vulnerability the security scheme may have. This is rarely a problem for Bootloaders, hence there is no way to observe more than a handful of encrypted images in the whole life time of the device. This often allows for softening of the requirements for the encryption algorithm, providing the benefits of smaller code size and increased performance. In this case, an attempt was made to follow best security practices, sacrificing some of the potential performance.

Security and authentication of the data is based on the master key stored in the first 16 bytes of the user area.

It is a good security practice to never use the master key directly, and derive a session key from the master key and other information specific to a given session or firmware image.

The procedure to obtain the session key is shown in the following example.

```
ObtainSessionKey(MasterKey, UnlockCommand)
    InitializeState();
    Absorb(MasterKey);
    Absorb(UnlockCommand);
    return Squezze(16)
```

The `MasterKey` is a 16-byte sequence from the user area, and the `UnlockCommand` is a full payload of the unlock command, excluding the command ID itself, but including the Guard Value. The length of this payload will be 28 bytes (Guard Value + Image Offset + Image Size + Nonce).

The resulting session key is unique to a specific image and location in the Flash memory. Nonce is used to eliminate possible duplication of the session keys, because typically application firmware images will be located at the same offset and have roughly the same size.

The session key is 16 bytes long, and it is used to initialize the encryption and authentication states for the individual blocks of data.

To generate a command payload for the `Data` command, raw data must be encrypted first. Encryption routine operates on blocks of data aligned at Erase Block Size boundary. It also takes command header (Guard Value and Block Offset) to ensure that the authenticated data cannot be written into a different location.

```
EncryptBlock(SessionKey, CommandHeader, Data)
    InitializeState();
    Absorb(SessionKey);
    Absorb('E');
    Absorb(CommandHeader);
    Absorb(Data);
    EncryptedData = Data + Squezze(EraseUnitSize)
    return CommandHeader || EncryptedData
```

Here `+` is a byte by byte arithmetic addition operation, and `||` is a concatenation.

After the session key, `EncryptBlock()` absorbs a single byte corresponding to the letter `E` (0x45). This is done to differentiate the internal state from the authentication operation. It is a good security practice not to reuse the same security material for encryption and authentication.

After the block of data is encrypted, it needs to be authenticated, since encryption is susceptible to ciphertext modifications that are not detectable during the decryption stage.

The authentication algorithm works on the block of the encrypted data returned from the `EncryptBlock()` function and it expands the encrypted data with a 16-byte authentication value.

```
AuthenticateBlock(SessionKey, EncryptedData)
    InitializeState();
    Absorb(SessionKey);
    Absorb('A');
    Absorb(EncryptedData);
    return EncryptedData || Squezze(16)
```

The final payload of the `Data` command accepted by the Bootloader will have the following structure: *CommandHeader (8 bytes)* || *EncryptedData (Erase Unit Size)* || *AuthenticationValue (16 bytes).*

7. Appendix B: PC Utilities for Working with Encrypted Images

A number of evaluation utilities are provided with the Bootloader. The process is split into preparing an encrypted image that can be freely distributed over open channels, and an actual update.

`encrypt.py` is an encryption utility. It has the following syntax:

Options:

```
-h, --help                    show this help message and exit
-f FILE, --file=FILE        input file
-k KEY, --key=KEY            encryption key (16 bytes separated with ':')
-o OFFS, --offset=OFFS         destination offset (default 0x800)
```

Example invocation:

```
python encrypt.py -k 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f -f
test_app_d10.bin
```

`encrypt.py` will add the suffix `.enc` to the input file name to get the name of the output file.

Each invocation will produce different file contents, because a random nonce is generated for each invocation.

There is a corresponding implementation in C called `encrypt.c`. It is absolutely equivalent to the Python version in syntax and operation. This version can be used as a reference for embedded application using the Bootloader and generating the image in run-time.

`boot.py` is an update utility, it takes an encrypted image and uploads it over the serial port. It has the following syntax:

Options:

```
-h, --help                    show this help message and exit
-v, --verbose               enable verbose output
-t, --tune                    auto-tune UART baudrate
-i PATH, --interface=PATH    communication interface
-f FILE, --file=FILE        binary file to program
--boot                        enable write to the bootloader area
```

Example invocation:

```
python boot.py -v -i COM12 -t -f test_app_d10.bin.enc
```

The `--boot` option is necessary if an encrypted image has offset less than the Bootloader size. This is an additional protection to prevent accidental overwrite of the Bootloader or user area.

`key_update.py` is a utility to generate key update file. This file can be used by the `boot.py` utility to perform the actual key update. It has the following syntax:

Options:

```
-h, --help                    show this help message and exit
-f FILE, --file=FILE        output file name
-k OLD_KEY, --key=OLD_KEY    old encryption key (16 bytes separated with ':')
-n NEW_KEY, --new=NEW_KEY    new encryption key (16 bytes separated with ':')
```

Example invocation (single line):

```
key_update.py -k 0:1:2:3:4:5:6:7:8:9:0:1:2:3:4:5 -n aa:bb:cc:dd:ee:ff:
00:11:22:33:44:55:66:77:88:99 -f new_key.enc
```

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.

Legal Notice

Trademarks

ISBN: 978-1-5224-2411-6

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

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

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

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

France - Saint Cloud
Tel: 33-1-30-60-70-00

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