2017 Embedded Capture-The-Flag (eCTF)



**Secure Firmware Distribution for Automotive Control**

**Secure Design Document**

UMass Lowell Team

**1. Secure Design .............................................................................................. 2**

**2. Implementation ........................................................................................... 9**

**3. Source Code Description.............................................................................. 11**

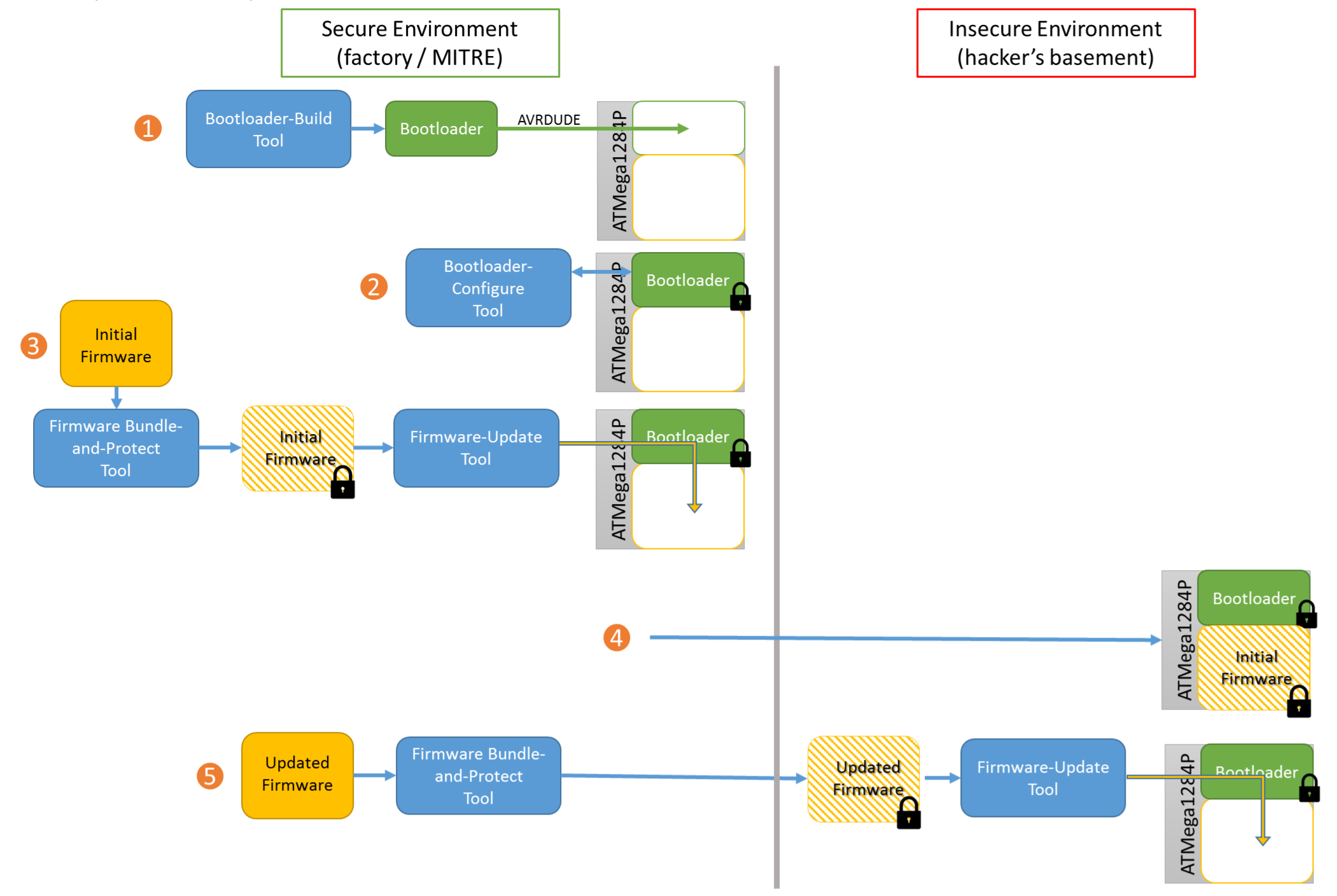
**1. Secure Design**

This chapter presents our system secure design. The secure design can satisfy all security goals and functional requirements.

**1.1 System Secure Design**

This section presents the design of a secure system which can support secure firmware distribution for automotive control and this design can reach the functional requirements.

*1.1.1 Secure System*



**Figure 1** System Use and Timeline

The Figure 1 shows the secure firmware distribution system use and timeline. This system has a secure bootloader which loaded on the ATMega1284p chip and five tools.

* Bootloader-Build Tool: It uses a compiler to build a series of .hex files that represent the bootloader and loads the .hex files onto chip via AVRDUDE.
* Bootloader-Configure Tool: It verifies that the bootloader was programmed successfully and saves all security parameters into an output file.
* Firmware Bundle-and-Protect Tool: It encrypts the original firmware and adds a supplied version number and a release message and a sha1 hash checksum to protect the firmware.
* Firmware-Update Tool: It installs a new protected firmware on target AVR chip.
* Readback Tool: It allows factory technician for debugging and accessing any region of flash memory.
* Secure Bootloader: It has three modes.
  + Load firmware mode: Communicate with Firmware-Update Tool and check the updated firmware integrity and authentication. Decrypt and load the firmware into flash program memory on chip after security checking.
  + Boot firmware mode: Print the release message and current firmware version on UART0 and boot the application firmware.
  + Readback mode: Communicate with Readback Tool and provide the requested data to Readback Tool after authority checking. The requested data is encrypted in order to protect communication sniff attack.

The secure design details are shown below.

*1.1.2 Bootloader-Build Tool*

This tool works inside the factory in a secure environment. It firstly generates a random secret password and a random cryptographic key and then uses them to compile the bootloader source code. It sets the fuse value and the bit lock value in order to protect software security and then loads the bootloader on target AVR chip via AURDUDE. The secret password and AES key will be saved to the output file “secret\_build\_output.txt”.

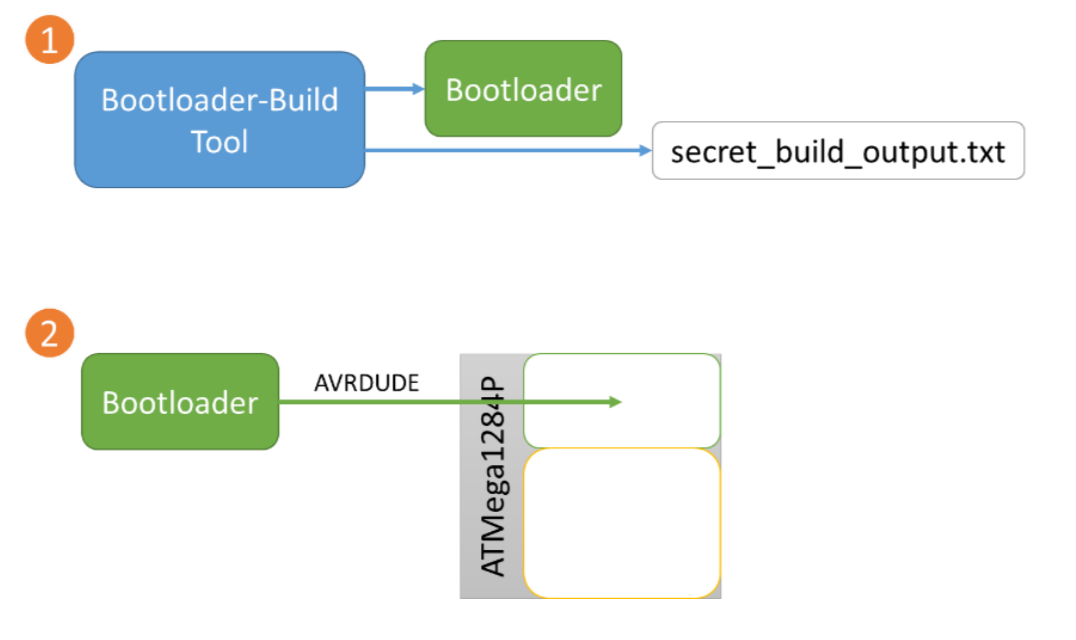


Figure 2 Bootloader-Build Tool

It has five steps.

1. Generate a random secret password (*password*) for readback tool and bootloader-configure tool. This value will be stored in secret\_build\_output.txt.
2. Generate a random AES encrypt/decrypt Key (*aeskey*) for firmware protect tool, readback tool and bootloader. This value will be stored in secret\_build\_output.txt.
3. Add the ‘*password’ and ‘aeskey’* as compiler flags “–DRB\_PASSWORD” and “-DAES” into the Makefile and compile the bootloader source code. This step generates the flash.hex, eeprom.hex and secret\_build\_output.txt files.
4. Copy the flash.hex and eeprom.hex into the host tools directory and write fues and Bit lock file. This step generates three fues files (lfues.hex, hfues.hex, efuse.hex) and a Bit lock file (lock.hex).
5. Use AVRDUDE for loading the bootloader and related .hex files on target chip ATMega1284p.

*1.1.3 Bootloader-Configure Tool*

This tool works inside the factory in a secure environment. It used for verifying that the bootloader was programmed successfully and saves all security parameters into an output file. In our design, we don’t need configuration post installation of the bootloader. So this tool is skipped.

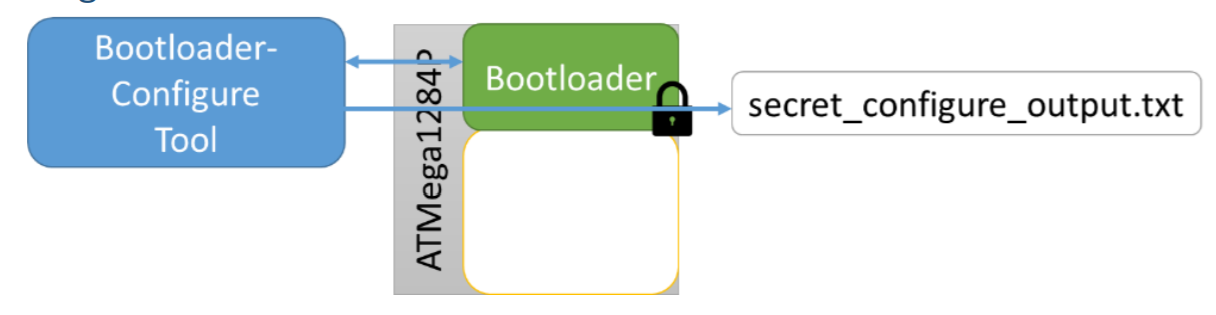


Figure 3 Bootloader-Configure Tool

It has one step.

1. Copy all security parameters from “secret\_build\_output.txt” file to the output file “secret\_configure\_output.txt”.

*1.1.4 Firmware Bundle-and-Protect Tool*

This tool works inside the factory in a secure environment. This tool uses AES encrypt method for encrypting the original firmware. Then it adds a supplied version number, a release message and a hash checksum to protect the firmware. We choose SHA1 hash function for firmware integrity and authentication checking. It consumes ‘secret\_build\_output.txt’.



Figure 4 Firmware Bundle-and-Protect Tool

It has five steps.

1. Check the original firmware size (*fw\_size*) cannot bigger than 30kB, version number (*version*) cannot bigger than 216 and releases message (*rl\_msg*) size cannot bigger than 1kB.
2. Get AES key (*KAES*) from “secret\_ build\_output.txt” and use it to encrypt the original firmware (*f.hex*).
   * Get encrypted firmware: *KAES(f.hex).*
3. Use SHA1 hash function (*Hsha1*) hash the encrypted firmware (*KAES(f.hex)*) and the release message. The hash value *Hsha1(KAES(f.hex) || rl\_msg)* can be used as the firmware id (*fid*) to protect the encrypted firmware and release message integrity.
   * Get firmware id: *fid* = *Hsha1(KAES(f.hex) || rl\_msg)*.
4. Use SHA1 hash function (*Hsha1*) hash the version number, firmware size, firmware id and AES key together. The hash value *Hsha1(version number || firmware size || firmware id || AES key)* can be used as the firmware checksum (*checksum*) to protect the firmware information.
   * Get firmware checksum: *checksum* = *Hsha1( version || fw\_size || fid || KAES ).*
5. Generate the secure firmware. The output secure firmware should have five parts.

'firmware\_size' : fw\_size,

'version' : version,

'hex\_data': *KAES(f.hex) || rl\_msg*,

'fid' : *Hsha1(KAES(f.hex) || rl\_msg)*,

'checksum': *Hsha1( version || fw\_size || fid || KAES )*

, where || means concatenation and ‘hex\_data’ has encrypted firmware and release message'.

*1.1.5 Firmware-Update Tool*

This tool works both inside and outside the factory. It cannot include any security parameters. This tool communicates with bootloader’s load firmware mode through UART1 and only installs the validated firmware on target AVR chip.

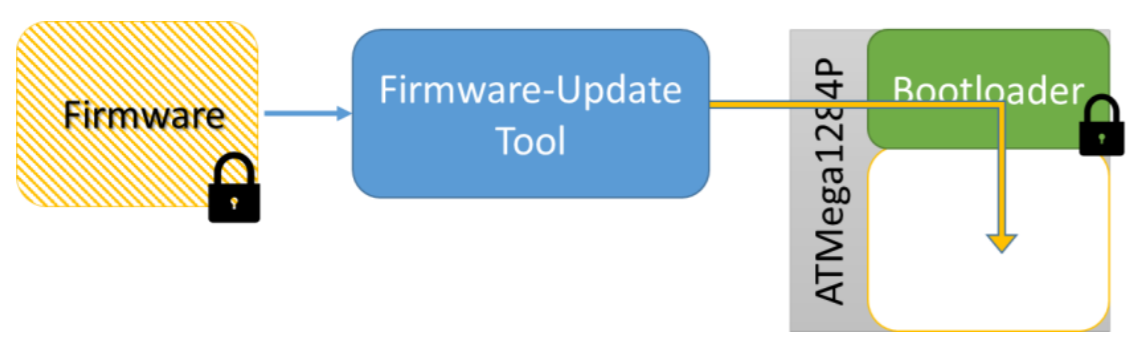


Figure 5 Firmware-Update Tool

It has four steps.

1. Send version (*version*), firmware size (*fw\_size*), firmware ID (*fid*) and firmware checksum (*checksum*) to bootloader.
2. Wait for bootloader integrity and authentication checking acknowledgment message.
3. Send encrypted firmware (*KAES(f.hex)* and release message (*rl\_msg)* to bootloader.
4. Wait for bootloader integrity and authentication checking acknowledgment message.

*1.1.6. Readback Tool*

This tool works inside the factory in a secure environment. It communicates with bootloader’s readback mode through UART1 and allows factory technician for debugging and accessing any region of flash memory. However, a factory technician accidentally released a recording of the readback tool onto the Internet. Therefore, it should be assumed that all attackers will have access to a recording of the communication from the bootloader to the readback tool.

In order to protect firmware, this tool extracts the secret password (*password*) and AES key (*KAES*) from “secret\_build\_output.txt” file. It will firstly provide secret password (*password*) to bootloader for authority checking. All requested firmware data will be encrypted before sending to readback tool. It decrypts the receiving data and print on screen.

*1.1.7 Secure Bootloader*

1. Load firmware mode

Put the jumper between pin PB2 and ground on protostack board and the bootloader will execute in load firmware mode. The mode communicates with Firmware-Update Tool and check the updated firmware integrity and authentication. It loads the firmware into flash program memory after security checking.

It has eight steps:

* Receive the metadata: version (*version*), firmware size (*fw\_size*), firmware ID (*fid*) and firmware checksum (*checksum*) provided by Update Tool.
* Get the AEK key (*KAES*) from EEPROM and calculate the sha1 hash (*Hsha1*) value of input metadata.
* Compare the *Hsha1*( *version* || *fw\_size* || *fid* || *KAES*) with the input firmware checksum (*checksum)* and send the acknowledgment response message to Update Tool.
  + OK: checksum is match
  + ERROR: checksum doesn’t match stop reading.
* If OK, read the encrypted firmware (*KAES(f.hex)* and releases message (*rl\_msg)* from Update Tool.
* Check input releases message size. If it is bigger than 1kB, send the acknowledgment message ‘ERROR’ to Update Tool and stop reading.
* Calculate the sha1 hash (*Hsha1*) value of input encrypted firmware and releases message.
* Compare the *Hsha1*(*KAES(f.hex)* || *rl\_msg*) with the input firmware id (*fid)* and send the acknowledgment response message to Update Tool.
  + OK: *fid* is match
  + ERROR: *fid* doesn’t match. Stop reading and erase the programed flash memory
* If OK, decrypt the input firmware (*KAES(f.hex)* and put it into flash program memory. Then send the acknowledgment message ‘OK’ to finish firmware update.

1. Boot firmware mode

Pull out the jumper on protostack board and the bootloader will execute in boot firmware mode. It will boot the application firmware and print the release message and current firmware version on UART0 in format: Msg: \*\*\*\*. Ver: 0002.

It has three steps:

* Get the release message from PROGMEM and current firmware version from EEPROM.
* Print these messages on UART0.
* Boot the application firmware.

1. Readback mode

Put the jumper between pin PB3 and ground on protostack board and the booloader will execute in readback mode. The mode communicates with Readback Tool and sends the requested data after authority checking. The requested data are encrypted in order to protect communication sniff attack.

It has eight steps:

* Receive the secret password (*PWT*) provided by Readback Tool.
* Get the original secret password (*PWC*) from PROGMEM and the AES key (*KAES*) from EEPROM.
* Compare two passwords and send the acknowledgment response message to Readback Tool.
  + OK: Passwords are match
  + ERROR: Passwords don’t match
* If OK, get and encrypt the requested firmware data from flash program memory and send it to Readback Tool.
* Wait for watchdog timer to reset.

**1.2 Security Goals**

This section presents our secure design can provide confidentiality as well as integrity and authentication checking and can reach the security goals.

*1.2.1 Confidentiality*

The attacker can capture the firmware data in two ways.

* New firmware release to the user

In this way, all attackers can get the same firmware. In our design, the original firmware has been protected by Firmware Bundle-and-Protect Tool before it releasing to the user. All original firmware has been encrypted by AES key and the firmware update tool sends the encrypted firmware to bootloader. The attacker doesn’t have AES key, thus he cannot reverse-engineering the application firmware.

* Readback recode on Internet

In this way, all attackers can access the recode and get the firmware data. In our design, the bootloader uses AES key to encrypt the requested firmware data and send it back to readback tool. The attacker doesn’t have AES key, thus he cannot reverse-engineering the application firmware.

*1.2.2 Integrity and Authentication*

The secure bootloader will check the validity of updated firmware. It will check the updated firmware integrity and authentication. All software access to the flash memory should go through the bootloader and the bootloader will check the authority for flash memory accessing.

* The bootloader checks the firmware checksum before receiving the encrypted firmware. The firmware checksum is the sha1 hash value of version number, firmware size, firmware id and AES key. The attacker doesn’t have AES key, so any change of these information will be detected in bootloader. And this checksum can authenticate the firmware id which is the sha1 hash value of encrypted firmware.
* The bootloader checks the firmware id after receiving whole encrypted firmware and release message. The firmware id is the sha1 hash value of the encrypted firmware and release message. Any change of the encrypted firmware or release message will be detected in this checking.
* The fuse and Bit Look setting can protect software security. Any software access to the flash memory should go through the secure bootloader and should provide the secret password for authority checking. Only the factory has the secret password.

Thus, any change of the updated firmware will be detected in bootloader update mode and the invalidity firmware will not be loaded in flash program memory.

**2. Implementation**

This chapter presents the implementation of our secure design and how to use of our secure bootloader and five host tools.

**2.1 Secure bootloader**

The secure bootloader is implemented in c language. All source code files are stored in ‘bootloader’ directory. The bootloader.c has been well commented inside the source code file and it contains four function source files. All source code files are stored in bootloader/src directory.

* The ase.c file implements the AES encrypt method.
* The sha1.c file implements the sha1 hash function.
* The sys\_startup.c file implements the basic system start up functions.
* The uart.c file implements UART port function.

All header files are stored in bootloader/include directory. It contains ase.h, sha1.h and uart.h header files. The Makefile are stored at the root level of bootloader directory. For more details, see the comment under 2017-ectf-master/bootloader/src /bootloader.c file.

**2.2 Host Tools**

All host tools are implemented in python language. Each tool has been well commented inside the source code file. All source code files are stored in ‘host\_tools’ directory. The use of these tools is described below. For more details, see each tool’s source code under 2017-ectf-master/host\_tools folder.

*2.2.1 Bootloader-Build Tool*

* Compile bootloader to generate the following files (which will be programmed onto the chip via avrdude):

1. flash.hex — The contents of flash memory.
2. eeprom.hex — The contents of EEPROM.
3. lfuse.hex — The contents of the lower fuse bits.
4. hfuse.hex — The contents of the upper fuse bits.
5. efuse.hex — The contents of the extended fuse bits.
6. lock.hex — The contents of the bit lock bits.

* Compile the bootloader, Cmdline: python bl\_build
* Write secret password and AES key to secret\_build\_output.txt.
* Load bootloader onto ATMega1284p, AVRDUDE cmdline:

avrdude -P usb -p m1284p -c dragon\_isp -B 200 -u -U flash:w:flash.hex:i -U eeprom:w:eeprom.hex:i -U lfuse:w:lfuse.hex:i -U hfuse:w:hfuse.hex:i -U efuse:w:efuse.hex:i -U lock:w:lock.hex:i

Or use avrdude.sh script, cmdline: sh avrdude.sh

*2.2.2 Bootloader-Configure Tool (\* optional: you can skip this tool)*

* Connect PC with UART1 port on protostack board.
* Cmdline: python bl\_configure –port <serial port>
* Copy all security parameters from secret\_build\_output.txt to secret\_configure\_output.txt.

*2.2.3 Firmware Bundle-and-Protect Tool*

* Cmdline: python fw\_protect –infile <unprotected\_firmware\_filename> --version <version\_number> --message <release\_message> --outfile < protected\_firmware\_filename >
* Consume secret\_build\_output.txt to protect and generate a secure firmware.

*2.2.4 Firmware-Update Tool*

* Connect PC with UART1 port on protostack board.
* Cmdline: python fw\_update –port <serial port> -- firmware <filename\_of\_protected\_firmware>
* Wait for an acknowledgement response from the bootloader.

*2.2.5 Readback Tool*

* Connect PC with UART1 port on protostack board.
* Cmdline: python readback –port <serial port> --address <start\_address> --num-bytes <number\_of\_bytes\_to\_read>
* Consume secret\_build\_output.txt to provide the secret password to bootloader.
* Wait for an acknowledgement response from the bootloader.
* Decrypt and print the requested data on screen.

**3. Source Code Folder Description**

At the root level of the 2017-ectf-master.zip file should be ‘Vagrantfile’ and three directories named ‘host\_tools’, ‘bootloader’ and ‘Firmware’.

The ‘Firmware’ directory contains two files:

* blinkLED.c: Application firmware source code.
* blinkLED.hex: Original Application firmware. It can be protected by Firmware Bundle-and-Protect Tool and be loaded into ATMega1284p through Firmware-Update Tool.

The ‘bootloader’ directory contains two folders and a Makefile:

* ‘include’ folder: It contains all function header files
* ‘src’ folder: It contains bootloader.c source code and all function source code.

The ‘host\_tools’ directory contains five tools source code and avrdude.sh script.

**3.1 Setup code test environment**

1. Download the 2017-ectf-master.zip source code folder.
2. Change into its directory in your shell and follow these steps to start up your VM:
   1. Copy your own Vagrantfile.local into this directory and ensure that the configuration make sense for your system configuration
   2. Run vagrant up
   3. Run vagrant ssh to log in to the VM
3. To connect the AVR Dragon to the protostack board, use the included ribbon cable to connect the 6-pin ISP header on the AVR Dragon to the ISP10 header on the protostack board.
4. To connect the PC to the protostack board, use the USB to RS232 cable to connect the RS232 pins on the protostack board UART ports.
5. Run the Tools

a. Change into ‘host\_tools’ directory and run command to compile the bootloader.

* + Cmd: python bl\_build

b. Load the bootloader and related .hex files on ATMega1284p.

* + Cmd: sh avrdude.sh

c. (\* optional) The bl\_configure tool will only copy all security parameters from secret\_build\_output.txt to secret\_configure\_output.txt.

* + Cmd: python bl\_configure –port <serial port>

d. Protect the original firmware.

* + Cmd: Python fw\_protect –infile <unprotected\_firmware\_filename> --version <version\_number> --message <release\_message> --outfile < protected\_firmware\_filename >

e. Put the jumper between pin PB2 and ground on protostack board, the bootloader will enter load firmware mode.

* + Cmd: python fw\_update –port <serial port> --firmware <filename\_of\_protected\_firmware>

f. Pull out the jumper on protostack board, the bootloader will enter boot firmware mode. The UART0 should print the release message and current firmware version on the screen. The message format is: “Msg: \*\*\*\*\*. Ver: 0002”

g. Put the jumper between pin PB3 and ground on protostack board, the bootloader will enter readback mode and run command to get requested data. The requested data will print on the screen.

* Cmd: python readback –port <serial port> --address <start\_address> --num-bytes <number\_of\_bytes\_to\_read>