University of Connecticut

MITRE eCTF

Brian Marquis, Patrick Dunham, Luke Malinowski, James Steel, Cameron Morris, Chenglu Jin, Waldemar Cruz, Paul Wortman
Dr. John Chandy
3/1/17

Contents

1 Memory Mapping

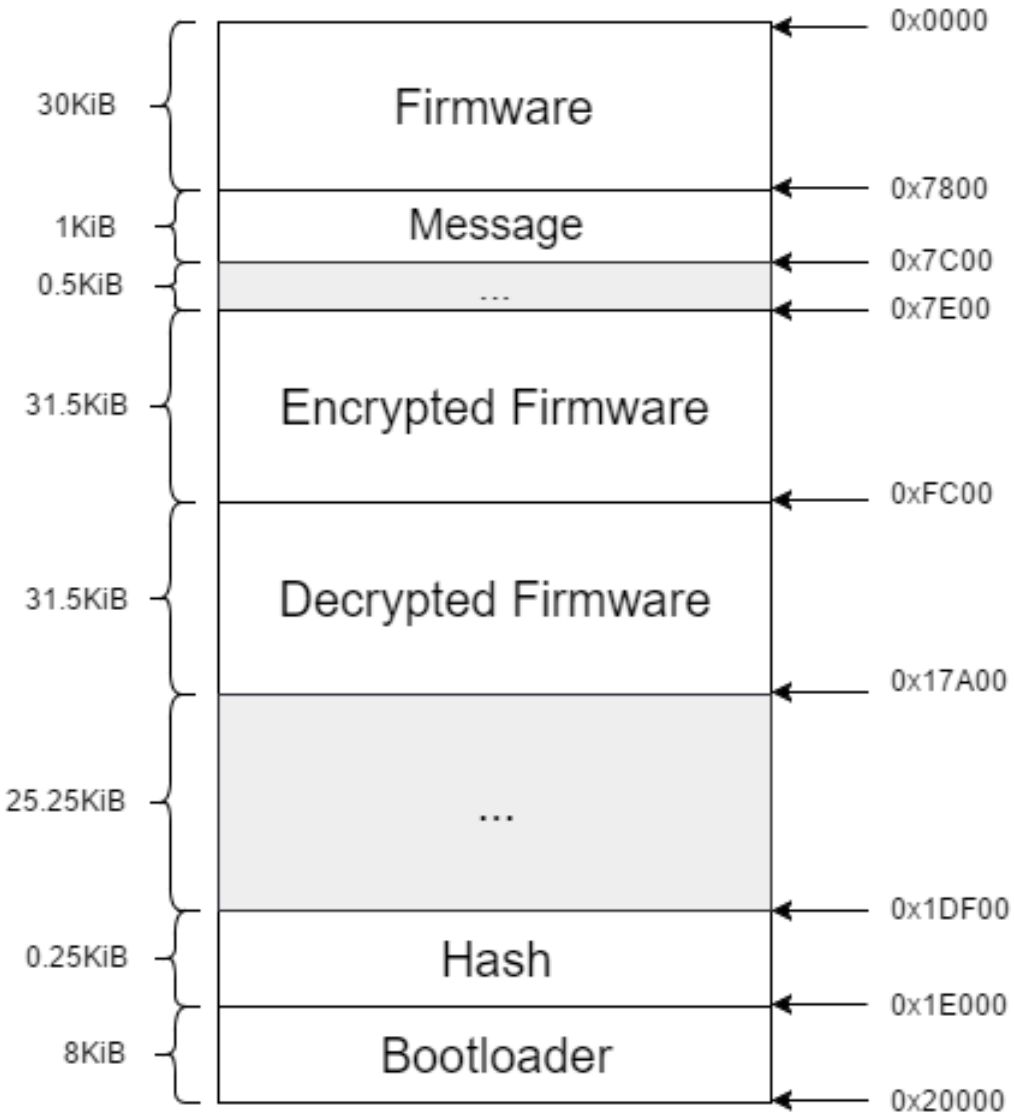

Figure 1: Flash Memory Mapping

2 Fuse Settings

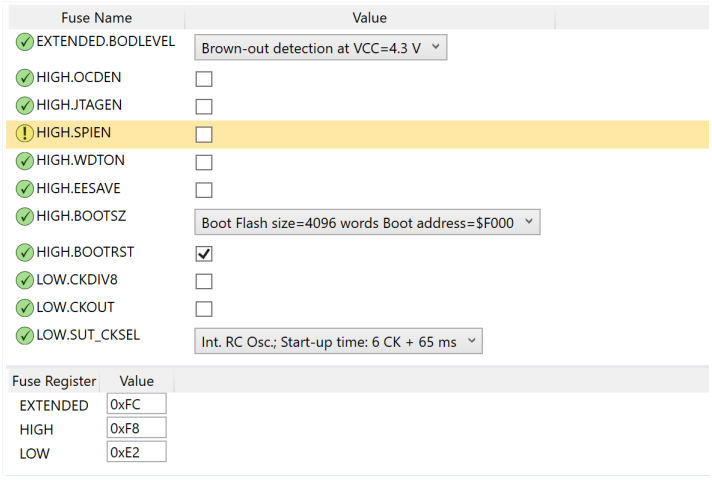

Figure 2: ATMega1284P Fuse Settings

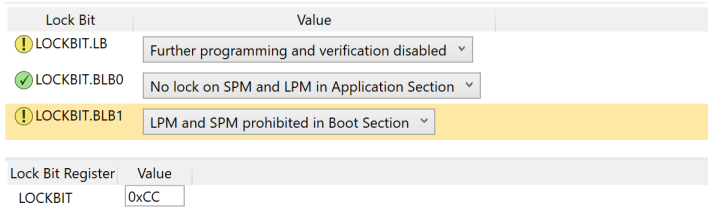

Figure 3: ATMega1284P Lock Bit Settings

3 Encryption

The bootloader makes use of AES-256 as the standard method of encryption throughout. It is extremely secure, fast, and relatively lightweight, making it an excellent choice for this application. In addition, its longevity and widespread use means that countermeasures to common attacks are widely available.

As AES-256 is a block cipher, it requires a mode of operation to properly operate on more than 16 bytes. For encryption and decryption, Cipher

Feedback Mode (CFB Mode) is the chosen method of operation. CFB Mode has a significantly smaller code size than other modes of operation, which is of extreme importance.

Each round of CFB Encryption encrypts the previous block of ciphertext and XORs it with the next block of plaintext. This produces the next block of ciphertext.

Equation: $Ciphertext[i] = AES(Ciphertext[i-1])XORPlaintext[i]$

For the first round, as there is no previous ciphertext, an Initialization Vector (IV) is used. This value is randomly generated and stored on the AVR, and is known to the host tools. Decrypting a message encoded with CFB mode operates in much the same way, with a few minor changes. A brief outline of this mode is presented in the diagram below:

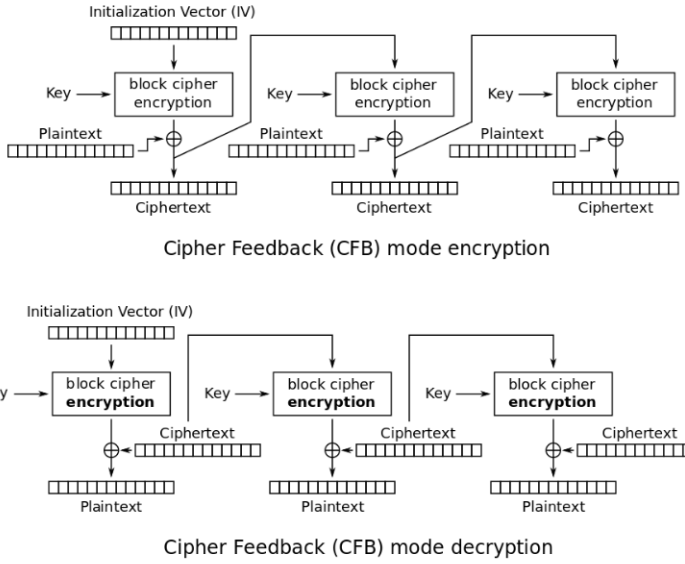

Figure 4: Cipher Feedback Diagram

Each round of CFB Decryption encrypts the previous block of ciphertext and XORs it with the next block of ciphertext. This produces the next block of plaintext.

Equation: $Plaintext[i] = AES(Ciphertext[i-1])XORCiphertext[i]$

Again, the IV is used during the first round, as there is no previous ciphertext. CFB Decryption is notable as it still uses the AES-256 block encryption method. Therefore, a significant amount of code can be reused, reducing code size.

The AES-256 encryption algorithom is used again for computing of hashes and Message Authentication Codes (MACs). This is accomplished through a slightly different block cipher mode of operation. The CBC-MAC algorithm is chosen, as unlike CMAC, it does not require generation of a second keyschedule, leading to smaller code size. CBC-MAC is as simple as running AES-256 in the Cipher Block Chaining Mode (CBC Mode), and only preserving the last encrypted block. An outline of this process can be viewed in the diagram below:

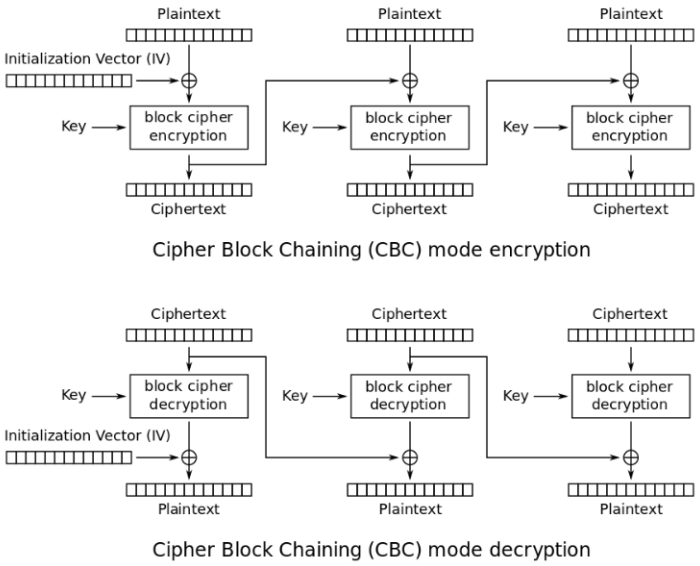

Figure 5: Cipher Block Chaining Diagram

Each block of the message is XORed with the previous MAC result, and is then encrypted.

Equation: $MAC[i] = AES(MAC[i-1]XORMessage[i])$

As with CFB mode, the first run of the MAC algorithm uses an IV.

Unlike other modes of operation, the IV for CBC-MAC is always 0. A separate key from the encrypted message is always used to generate the MAC, and fixed-length messages are used throughout. This eliminates the commonly known weaknesses of CBC-MAC.

4 Bootloader

4.1 Configure Bootloader

When the bootloader is installed a flag is set in EEPROM to show that it has not yet been configured. In the main function, if that flag is set, the configuration will occur. First, the internal RC oscillator will need to be calibrated to yield a frequency of about $7.37MHz$ so that a baudrate of 115200 can be used on UART. To do this, a binary search algorithm is used to determine whether the current OSCCAL value is too high or low. The algorithm uses an external interrupt and Timer 0 to measure the UART pulse width and compares that to the value it ought to be for 115200. This feedback allows the binary search to find the proper OSCCAL value for the chip. That value is then written to EEPROM so that it can be pulled on boot.

The bootloader will read each page of flash memory dedicated to the bootloader. Each page of bootloader flash will also be iterated through a hash function. Once each page is read and the final hash is generated, this value will be sent over UART1. The tool will then check to see if the received value matches the hash calculated by the computer. If it matches, the tool will send an ACK over UART1. Otherwise, it will send a NACK (0x15). If a NACK is received, the bootloader will set a flag indicating that either the bootloader or EEPROM has been installed incorrectly. Then, it will wait for the Watchdog timer to reset the system. If ACK is received, on the other hand, the bootloader will proceed to read the EEPROM section one page at a time. Like before, the bootloader will generate a hash on this data and will send it to the configure tool over UART1. As before, if the hash is correct, the tools will send an ACK; otherwise, it will send a NACK. If an ACK is received by the bootloader, the system will wait for a Watchdog reset. Otherwise, it will set the configuration error flag and then wait for the reset.

4.2 Readback Flash

When a jumper to PB3 is connected, the bootloader will send an R and enter the readback function. The function will wait for the tools to send the starting flash address, the readback size, the password, and a hash of the starting address and the readback size. The bootloader will then compute the hash of the encrypted starting address and size and compare that against the hash sent by the tools. If incorrect, the bootloader will not send any flash data over UART1 and will reset with the Watchdog timer. Otherwise, the bootloader will proceed to decrypt the encrypted received data. Once decryption is complete, the decrypted password will be checked against the password stored in EEPROM. If valid, the bootloader will use the decrypted size and starting address to send each page (256 bytes) of encrypted flash memory over UART1. The bootloader will send as many pages as needed to transmit the requested amount of data. Then, the Watchdog timer will reset the bootloader.

4.3 Load Firmware

```
 * \brief Loads a new firmware image and release message
 *
 * This function securely loads a new firmware image onto
 * flash, following the procedure outlined below.
 *
 *
 * 1 - The encrypted firmware image is loaded into the
 *      ENCRYPTED_SECTION of flash.
 * 2 - The CBC-MAC of the encrypted firmware image is
 *      computed, and compared to the CBC-MAC sent.
 *      IF   CORRECT -
 *        The bootloader proceeds with the firmware upload.
 *      IF INCORRECT -
 *        The bootloader erases ENCRYPTED_SECTION and terminates.
 * 3 - The firmware image is decrypted and stored in the
 *      DECRYPTED_SECTION of flash.
 * 4 - The version number is checked versus the current version,
 *     and updated in EEPROM
 * IF   CORRECT -
 *       The bootloader proceeds with the firmware upload.
 * IF INCORRECT -
```

```
*       The bootloader erases ENCRYPTED_SECTION and
*  DECRYPTED_SECTION and terminate.
* 5 - The release message is written to the MESSAGE_SECTION
* 6 - The firmware is written to the APPLICATION_SECTION
* 7 - The bootloader erases ENCRYPTED_SECTION and
*      DECRYPTED_SECTION and terminates
```

4.4 Boot Flash

If the firmware verification flag is not set when the boot function is called, then the bootloader will attempt to jump to address zero and begin executing the first instruction in the application flash.

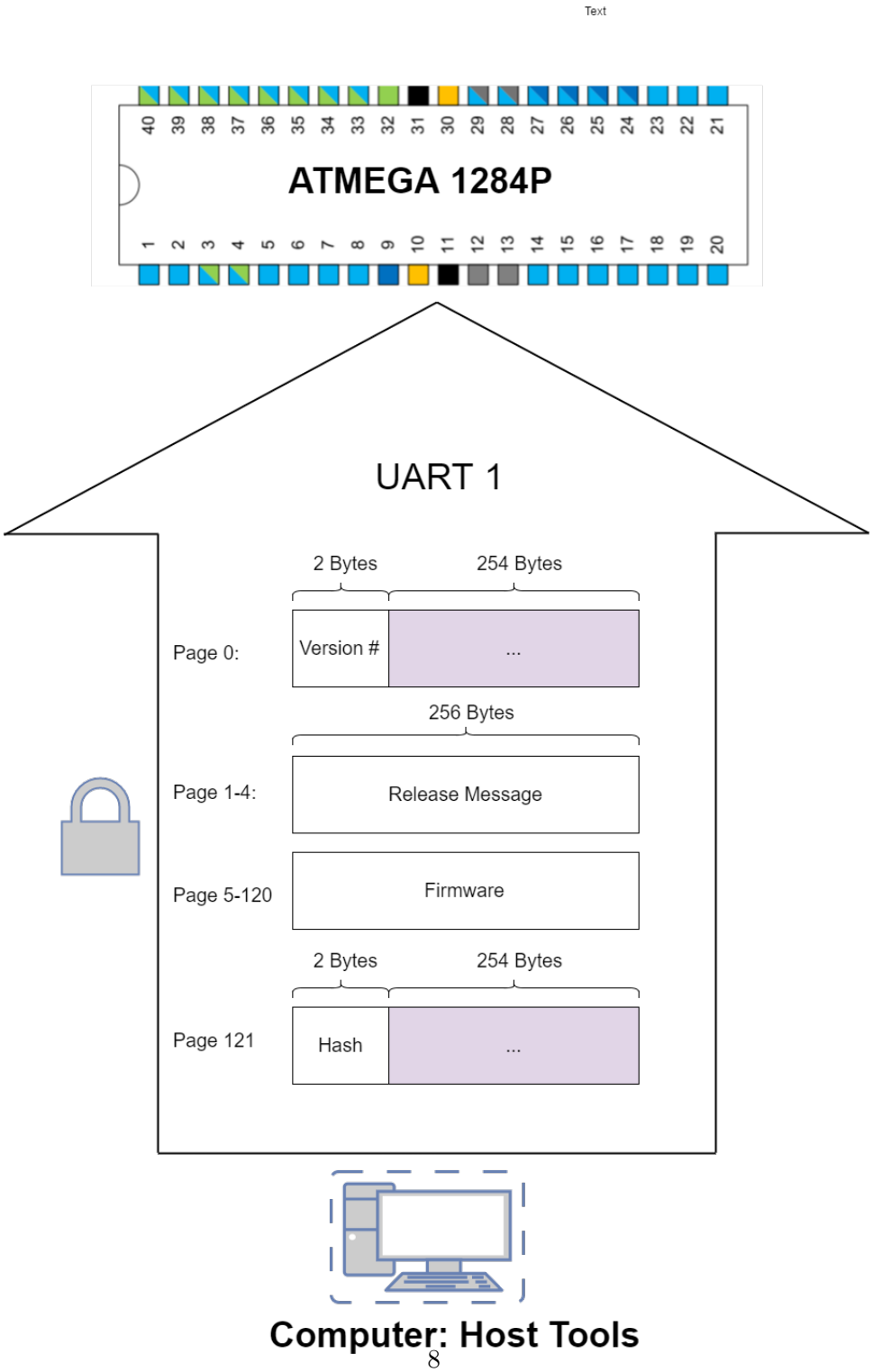

Figure 6: Load Firmware Packet Orientation

5 Host Tools

The host tools consist of bl_build, bl_config, fw_update, fw_protect, and readback.

5.1 Build Tool

The bootloader is built and written to an intel hex file. Secrets (keys, initialization vectors, readback password) are generated using /dev/urandom, and are written to secret_build_output.txt. The build tool constructs the flash memory of the bootloader then hashes it with CBCMAC and stores the hash in the secrets file

5.2 Configure Tool

The configure tool verifies that the bootloader and EEPROM was written correctly by asking the AVR to compute a hash of each and comparing them to the hashes generated using bl _build. If the hashes do not match, the tool returns NACK and informs the user. If the hashes match the tool returns ACK and informs the user that the bootloader has installed correctly.

5.3 Readback Tool

The readback tool communicates with the AVR over an encrypted UART channel in order to read back a section of flash memory. It reads secrets from the secret_configure _output.txt generated by bl_build.py and constructs a request to read n bytes of memory starting at address a to send to the AVR. It then expects an encrypted response containing those requested bytes. After receiving all of the expected bytes, the readback tool will decrypt them and write them to stdout as hex.

5.4 Bundle and Protect Tool

Firmware is protected by encrypting and hashing it. AES-256 is used to encrypt the firmware, version number, and message. The encrypted blob is hashed using CBCMAC so the bootloader can verify it has not been modified.

5.5 Firmware Update Tool

The firmware update tool uploads the firmware to the AVR, one encrypted chunk at a time. It periodically checks to ensure that the AVR is

responding with ACK (0x06) on the same UART interface that the firmware is being uploaded through. If ACK is not received, then an error occurs.