Contents

[1. Secure Bootloader Design 2](#_Toc481615155)

[1.1 Overview 2](#_Toc481615156)

[1.2 Hardware 3](#_Toc481615157)

[1.3 Cryptographic Algorithms 4](#_Toc481615158)

[1.3.1 Encryption 4](#_Toc481615159)

[1.3.2 Decryption 5](#_Toc481615160)

[1.3.3 MAC 5](#_Toc481615161)

[1.4 Memory Map 6](#_Toc481615162)

[1.5 Modes of Operation 8](#_Toc481615163)

[1.5.1 Configure 8](#_Toc481615164)

[1.5.2 Firmware Upload 8](#_Toc481615165)

[1.5.3 Readback 9](#_Toc481615166)

[1.5.4 Boot 10](#_Toc481615167)

[1.6 Countermeasures 11](#_Toc481615168)

[1.6.1 Clock Attack 11](#_Toc481615169)

[1.6.2 Brownout Attack 11](#_Toc481615170)

[1.6.3 Power Side Channel Attack 12](#_Toc481615171)

# Secure Bootloader Design

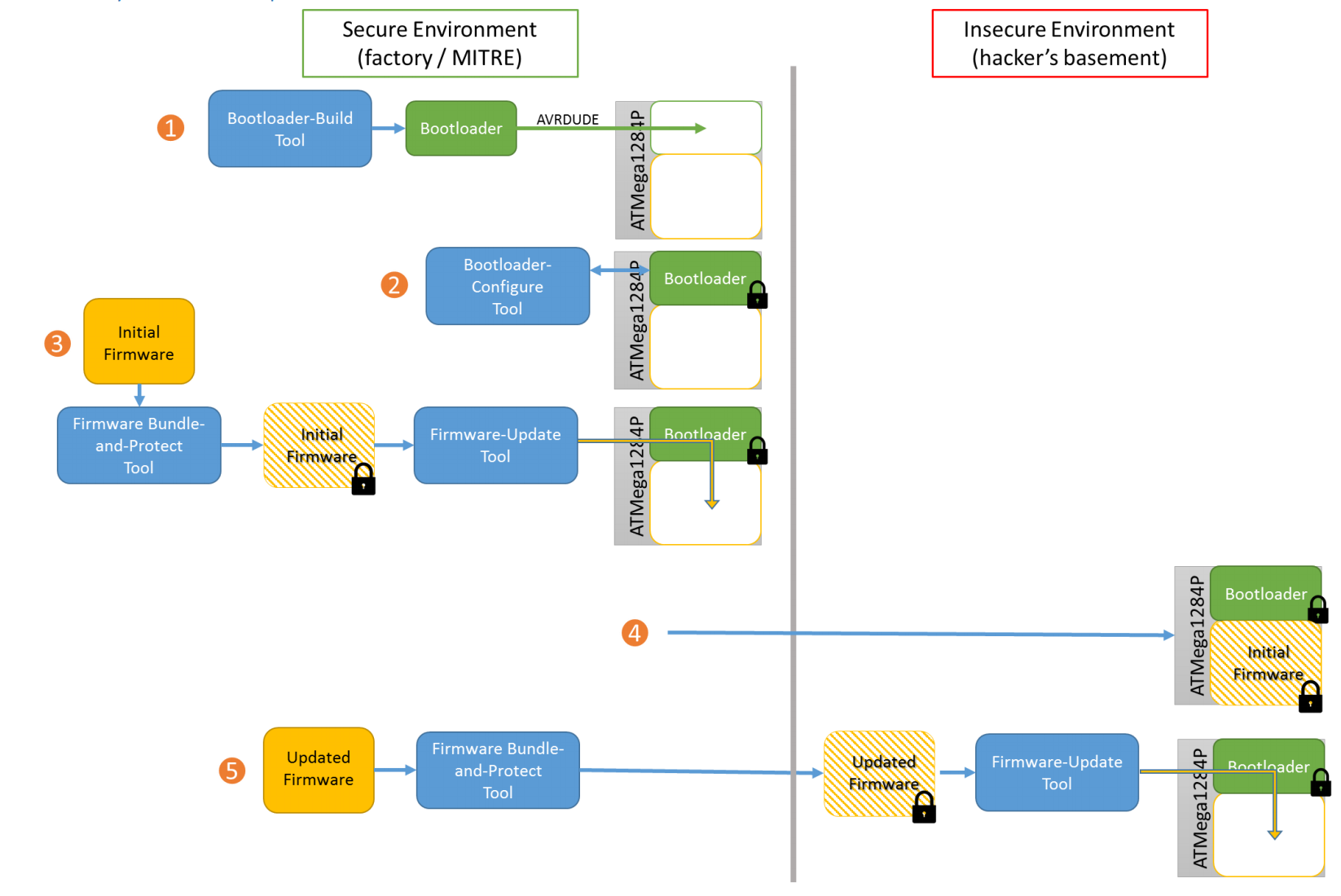
## Overview

The bootloader created by the UConn team for the MITRE eCTF event is designed to meet a minimal set of requirements, as detailed in the competition description. These requirements can be viewed below:

* Bootloader must be capable of uploading new firmware images up to 30 KiB in size
* Bootloader must support printing release messages up to 1 KiB in size
* Bootloader must be capable of reading back sections of installed firmware
* Bootloader must support a version control system
  + Only firmware of an equal or higher version number can be uploaded
  + Firmware Version 0 can always be uploaded for debug purposes

In addition to these basic requirements, it was required that the bootloader and firmware images be secure, preventing any unauthorized access. This way, the designed bootloader can be used in safety-critical environments, such as the electronic control unit of an autonomous car. Securing the bootloader and firmware images prevents attackers from gaining control of such devices, protecting the lives of those involved. In addition, securing the firmware images prevents attackers from stealing valuable intellectual property.

A basic design meeting the functional requirements was provided by MITRE, displaying how all bootloader components were to work together. While the design satisfied all functional requirements, it was not secured in any way. A view of how the design interacts with the various host tools can be viewed in Figure 1.1 – 1.

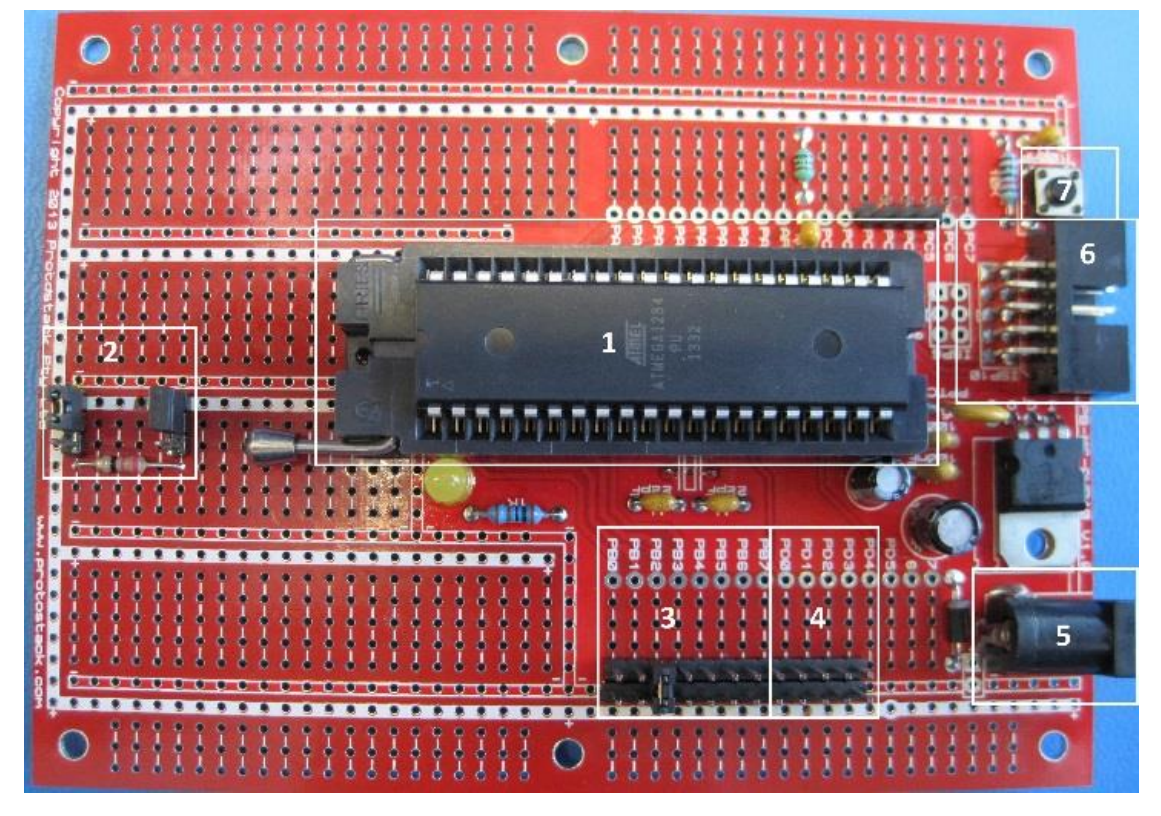


*Figure 1.1 – 1: MITRE Bootloader Functional Requirements (from MITRE)*

The UCONN team was allowed to add, modify, or rewrite this example bootloader, but the core connection between the host tools and bootloader had to be maintained.

## Hardware

The MITRE eCTF event required the use of specific hardware. All bootloaders were required to function on Atmel’s ATMega1284P 8-bit microcontroller installed in a supplied Protostack development board, with no modifications. This was to ensure that bootloaders designed by multiple teams would always function, regardless of external hardware. The supplied development board, pictured in Figure 1.2 – 1, consisted of a ZIF (Zero Insertion Force) socket containing the ATMega1284P [1], an optional power sense resistor [2], male headers for Port B GPIO (General Purpose Input – Output pins) [3], male headers for UARTs (Universal Asynchronous Receiver – Transmitter) 0 and 1 [4], a 5 V voltage regulator and barrel jack connector [5], an ISP (In – circuit Serial Programming) header [6], a reset button [7], and a 20 MHz external crystal.

*Figure 1.2 – 1: Protostack Development Board (from MITRE)*

UART 1 was reserved for host tool communication, while UART 0 was reserved for debug messages and general use, including release message printing. Pin B2 was used to signal entrance of Upload Mode, while Pin B3 was used to signal the entrance of Readback Mode. No other signals were allowed to be used during bootloader operation.

The ATMega1284 chip used has a total of 128 KiB of Flash memory, 4 KiB of EEPROM (Electrically Erasable Programmable Read-Only Memory), and 16 KiB of SRAM (Static Random Access Memory) available. In addition to the total physical memory restrictions, only 8 KiB of Flash can be used for bootloader space. These tight space restrictions mean that both total program size and RAM usage need to be monitored closely. While the hardware available was quite limited, this allowed the secure bootloader design to have a very small attack surface, ensuring that the resulting design is hardened against a large number of attacks.

## Cryptographic Algorithms

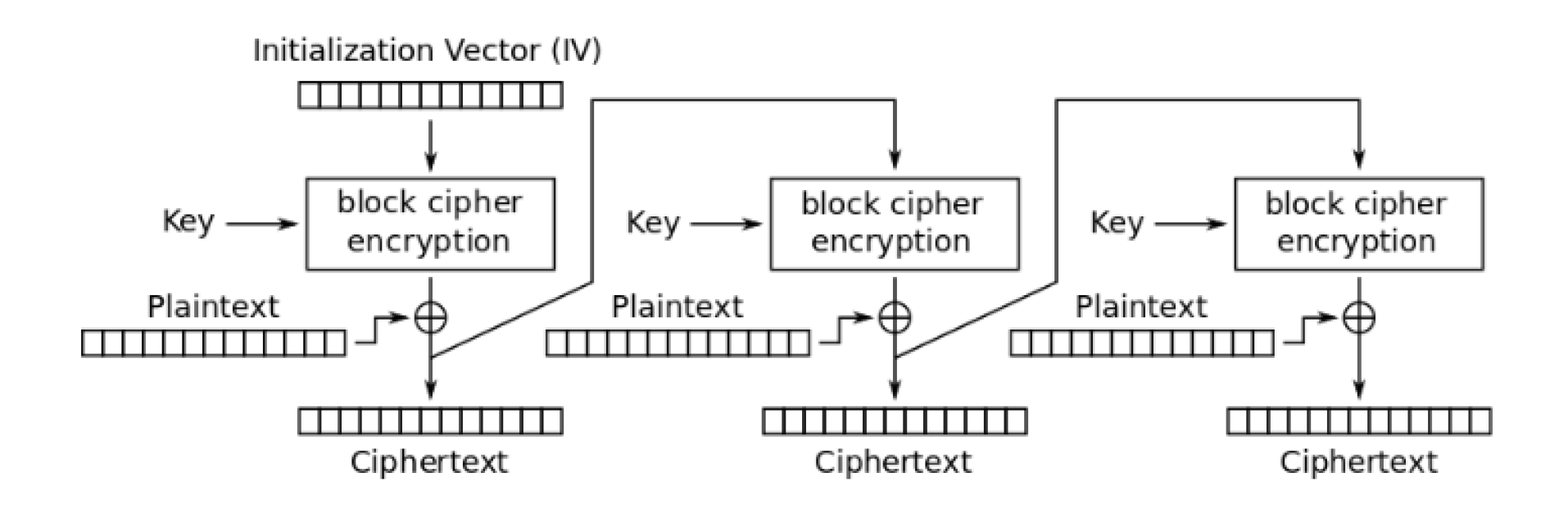
The final secure bootloader design is required to encrypt, decrypt, and calculate the MAC (Message Authentication Code) of various pieces of data. All three processes are based on the widely studied and used AES-256 encryption algorithm, a block cipher that operates on 16-byte blocks with a 256-byte key. This encryption core was chosen because of its proven security, its small size, and its low execution time. By using the AES *encryption* core for both encryption and decryption, a significant reduction in program size is achieved.

To prevent catastrophic failure if the encryption is recovered, encryption, decryption, and MAC generation all use different keys and initialization vectors. This way, if one process is compromised, the rest still function perfectly.

### Encryption

The encryption algorithm used in the secure bootloader design is AES-256 in CFB (Cipher FeedBack) mode. The primary motivator in this decision was the fact that CFB mode uses the block cipher *encryption* core for both encryption and decryption. In addition, as a block chaining mode, the ciphertext is based on both the AES key and previous ciphertext, further obscuring the message from attackers.

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. This process can be viewed in Figure 1.3 – 1.

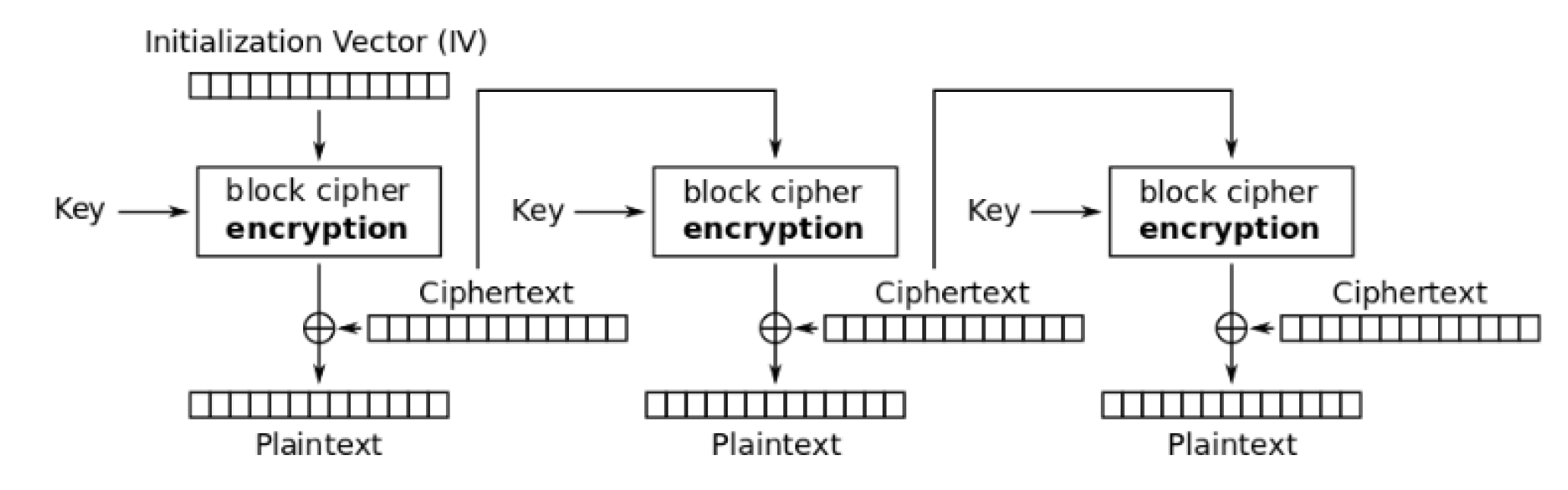


*Figure 1.3 – 1: CFB Mode Encryption (from Wikipedia)*

For the first round, as there is no previous ciphertext, an IV (Initialization Vector) is used. This value is randomly generated and stored in the ATMega1284P, and is known to the host tools.

### Decryption

As with encryption, messages are decrypted by the bootloader using AES-256 in CFB Mode. Decrypting a message encoded with CFB Mode operates in much the same way, with a few minor changes. 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. This process can be viewed in Figure 1.3 – 2.

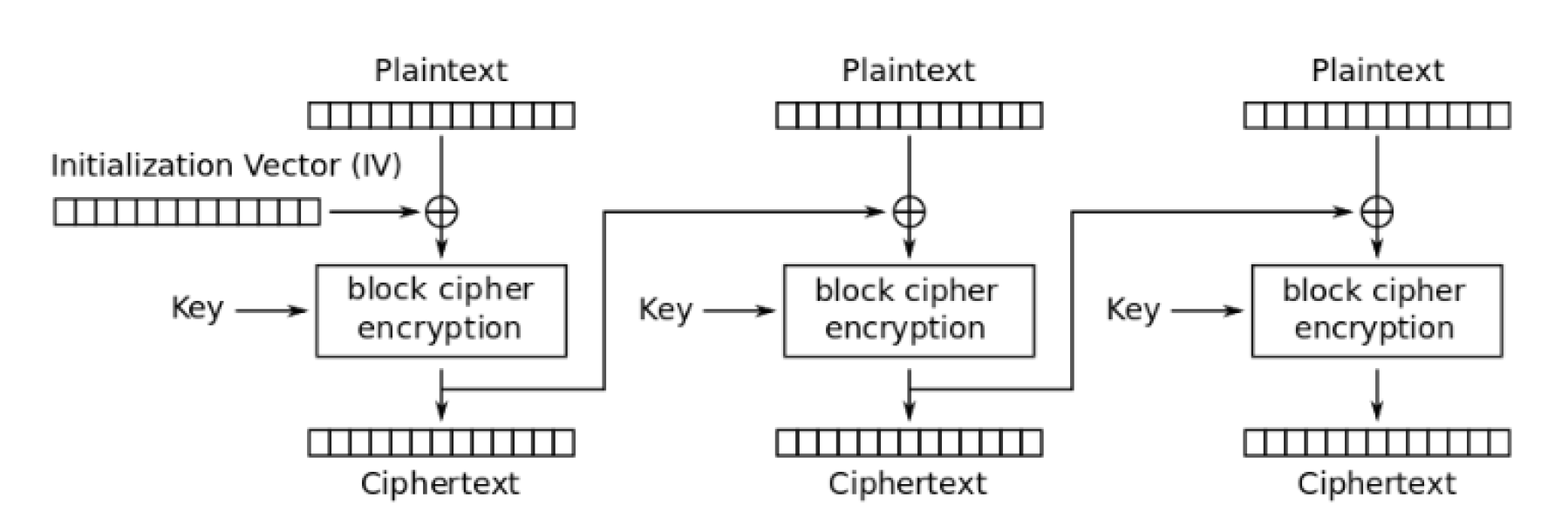


*Figure 1.3 – 1: CFB Mode Encryption (from Wikipedia)*

For the first round, as there is no previous ciphertext, an IV (Initialization Vector) is used. This value is randomly generated and stored in the ATMega1284P, and is known to the host tools.

### MAC

MAC generation in the bootloader uses CBC – MAC, which takes advantage of a block cipher running in CBC (Cipher Block Chaining) mode. As with encryption and decryption, the AES-256 algorithm is used. Unlike CMAC, another method of using a block cipher as a MAC generator, CBC – MAC does not require the generation of another keyschedule outside of AES, making it simpler and smaller to implement. CBC – MAC is as simple as running AES-256 in CBC Mode, and only preserving the last encrypted block. Each block of the message is XORED with the previous MAC result, and is then encrypted. This process can be viewed in Figure 1.3 – 3.



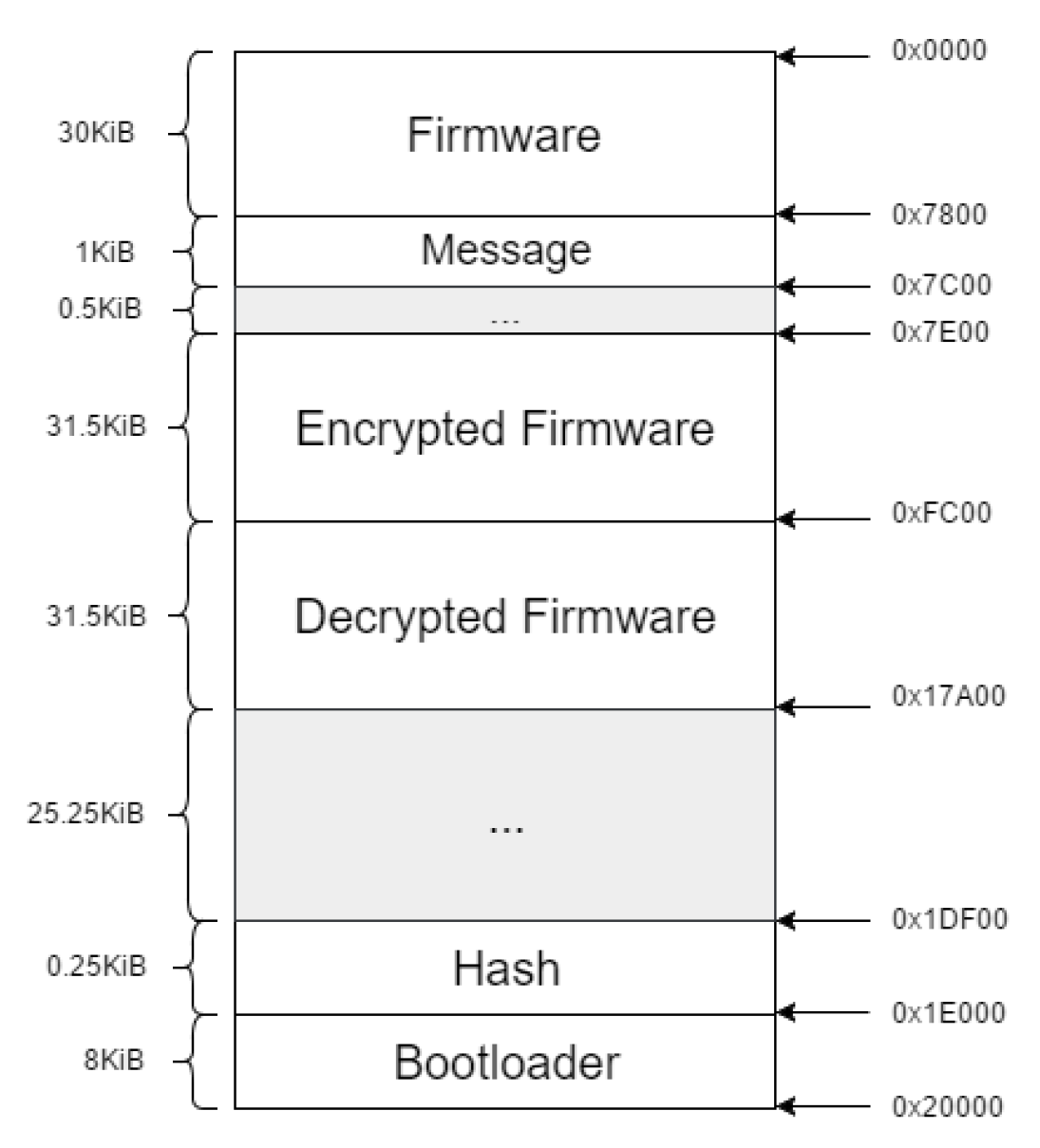
*Figure 1.3 – 3: CBC – MAC Generation (from Wikipedia)*

CBC – MAC requires an IV of all zeroes, unlike the encryption and decryption modes.

While CBC – MAC has known security flaws, these are easily solved by using a separate key from the encrypted message, and by only sending fixed-length messages to the ATMega1284P. These simple fixes result in a cryptographically secure MAC generation algorithm.

## Memory Map

The Flash memory of the ATMega1284P is segmented into a number of sections as detailed in Figure 1.4 – 1.



*Figure 1.4 – 1: ATMega1284P Memory Map*

The application section is divided into three 31.5 KiB sections, each of which store a firmware image in a different stage of decryption. The “Encrypted Firmware” section stores the original, unmodified message as sent by the host tools. The “Decrypted Firmware” section stores the decrypted firmware before the version of the firmware is verified. The first 31.5 KiB is further divided into two smaller sections. The 30 KiB section at the start of the memory is reserved for the valid, installed firmware image. The 1 KiB section is reserved for the valid, decrypted release message. The remaining 0.5 KiB is reserved for future purposes, such as storing various bits of metadata. An additional 25.25 KiB is reserved at the end of the application section. The final “Hash” section, while reserved and named, is not used, and intended for a future bootloader upgrade which will allow verification of installed firmware images. The final 8 KiB “Bootloader” section is used to store the entirety of the secure bootloader.

## Modes of Operation

The secure bootloader is designed to operate in one of four different modes, chosen at the start of the bootloader runtime. The four modes are listed below:

* Configure Mode
* Firmware Upload Mode
* Readback Mode
* Boot Mode

Each mode will be described in detail in the following sections.

## Configure

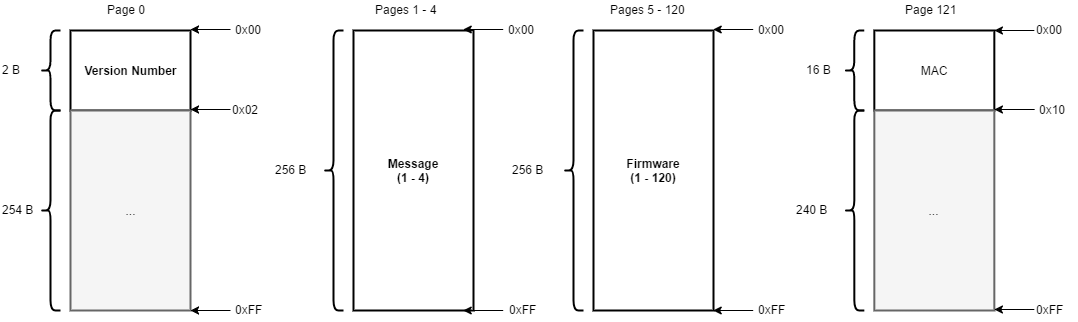
The bootloader enters Configure Mode after being installed for the first time, and will only ever be run once. Configure Mode is designed to accomplish two things: the calibration of the internal clock, and the verification of the installed bootloader. First, the internal clock is calibrated. This is done by using UART pulses generated by the host tool, as described in application note AVR054. As these UART pulses take a known length of time, they can be used as a reference clock for the ATMega1284P. The clock calibration register values correlate linearly with the internal clock frequency, allowing a binary search to be performed. The ATMega1284P first uses an external interrupt pin located on the UART 1 RX (receive) line and a timer to measure the pulse length in instruction cycles. It then compares it to the expected number of instruction cycles, and adjusts the clock calibration register by a set amount. This adjust amount is then reduced by a factor of 2, and the process is repeated a total of 5 times. The final clock speed is approximately 7.37 MHz, allowing for accurate baud rate generation of rates up to 115200 baud.

Once the clock is correctly configured, true serial communication can take place. After informing the host tool calibration has succeeded, and receiving a confirmation, the bootloader calculates a MAC of itself, and sends the MAC to the host tool. The host tool compares this against its own known reference, and informs the factory that the chip is correctly configured and the bootloader is properly installed. At this point, the clock calibration value is stored in EEPROM. In addition, an EEPROM flag is set informing the bootloader that the configuration step has been successfully completed.

## Firmware Upload

The bootloader enters Firmware Upload Mode when a jumper is attached to pin B2, pulling it to ground. This voltage level check is performed only at the start of the bootloader, typically requiring the ATMega1284P to be restarted after the jumper is connected. Firmware Upload Mode is designed to allow users to upload new firmware images without factory assistance, and to allow technicians to upload debug firmware to check product functionality. As the integrity and validity of the installed firmware determine the safety of the end users, it is essential to ensure only valid, tested firmware images are installed. In addition, as the firmware images represent valuable intellectual property, it is essential that firmware images be encrypted.

Upon entering Firmware Upload Mode, the bootloader transmits ASCII character ‘U’, and waits for a response from the host tool. If the host tool does not respond, the bootloader times out after 2 seconds and restarts. If the host tool responds, the 31.5 KiB firmware update message is received and analyzed. The firmware update message is divided into 122 pages, each consisting of 256 bytes of information. The first page is devoted to the firmware version number. The next 4 pages are devoted to the 1 KiB release message. The final page contains the calculated MAC of the message. The other 115 pages are devoted to the 30 KiB firmware image. Any page that does not fill the full 256 byte page size is padded with random bytes. The message structure can be viewed in Figure 1.5 – 1.



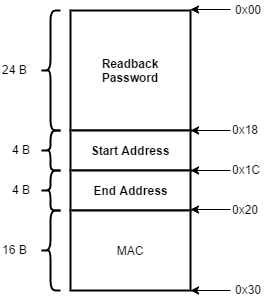
*Figure 1.5 – 1: Firmware Upload Message Structure*

First, each page is read in and stored in the “Encrypted Firmware” section of Flash. Once that is complete, the MAC is calculated and compared to the sent MAC. If there are no errors, the bootloader begins decrypting the firmware page by page, and stores the firmware in the “Decrypted Firmware” section. Once complete, the version number is compared to the stored version number. If the version number is greater or equal to the stored number, the stored number is updated and the firmware is installed. If the version number is 0, the firmware is installed without updating the version number. The decrypted, verified firmware image is placed in the “Firmware” section, and the message is placed in the “Release Message” section. Once this process is complete, the the ATMega1284P erases all used resources and restarts.

## Readback

The bootloader enters Readback Mode when a jumper is attached to pin B3, pulling it to ground. This voltage level check is performed only at the start of the bootloader, typically requiring the ATMega1284P to be restarted after the jumper is connected. Readback Mode is designed to allow a field technician or factory employee to troubleshoot installed firmware, verifying its integrity, version, and operation. As it is possible that the readback communications can be monitored, it is essential that the transmissions from the bootloader be secure and encrypted.

Upon entering Readback Mode, the bootloader transmits ASCII character ‘R’, and waits for a response from the host tool. If the host tool does not respond, the bootloader times out after 2 seconds and restarts. If the host tool responds, the 48-byte readback request is received and analyzed. The readback request is broken into 2 main sections: the message and the MAC. The message section consists of a 24-byte readback password followed by a 4-byte start address and a 4-byte end address. Figure 1.5 – 2 displays the message format, with encrypted sections in bold.



*Figure 1.5 – 2: Readback Request Structure*

The message is purposefully designed so that it aligns perfectly with the 16 byte block size of AES-256. Once received, the MAC of the message is calculated, and compared to the MAC sent with the message. If the comparison is valid, the message is decrypted. Once the message is decrypted, the readback password is compared to the stored readback password, if correct, the readback request is accepted.

The accepted readback request addresses are then page-aligned, to be compatible with the functions used to read and write from Flash used in the Firmware Upload Mode. Therefore, regardless of the number of bytes requested, the number of bytes returned is always a multiple of 256. This, in addition to reducing code size by reusing functions, ensures that the readback response is aligned with the 16 byte AES block size. It is ensured that at least one page of Flash is returned for any given readback message. Before reading Flash, it is checked that the pages requested originate from the “Firmware” section of memory. If they are not, the addresses are truncated, and nothing beyond the last page of flash is returned. Each page of Flash is encrypted, and sent to the host tools in order. Once the requested Flash section has been returned, the ATMega1284P resets.

## Boot

The bootloader is required to properly boot into installed firmware. When no other mode is requested, the bootloader automatically enters Boot Mode. During this mode, it deallocates all resources used by the bootloader, prints the release message over UART 0, and makes the jump to the application section.

It was originally intended that the bootloader also calculate a MAC of the installed firmware, and compare it against a saved MAC received when the firmware image was installed. This would have potentially allowed detection of malicious or modified firmware, and could have uninstalled dangerous applications. However, the current implementation of CBC – MAC relies on the provided side-channel-resistant AES-256 encryption core, which takes roughly 120 seconds to execute. It was determined that this delay on boot would be unacceptable. Improvements in the performance of the side-channel-resistant AES code would allow this feature to be implemented. As the memory space is already allocated, minimal modification to the source code would be required.

## Countermeasures

When designing the secure bootloader, a number of attacks were considered. The following attacks were determined to be the most likely to occur, and therefore had specific countermeasures designed to neutralize or mitigate the risk of them occurring:

* Clock Attacks
* Brownout Attacks
* Power Side Channel Attacks

While other attacks, such as chip decapping, were possible, they were deemed either too expensive or too time-consuming to reasonably occur.

In addition to the countermeasures described below, the ATMega1284P’s lockbits were set, disabling all programming and verification of on-chip Flash and EEPROM. In addition, both ISP and JTAG were disabled, making it even more difficult to communicate with the chip in any non-secured fashion.

## Clock Attack

One of the most devastating attacks considered was the clock attack. A kind of fault attack, the clock attack relies on manipulation of the MCU clock to cause glitches, or skip instructions. During the attack phase, it was found that this attack managed to skip a wide variety of instructions with more than 50% reliability, using only minimal setup. Because of the power of such an attack, preventing this was top priority.

The solution for this problem was very straightforward. Atmel’s AVR chips all have access to a low-power internal RC oscillator that can function as the system clock. By using this as the system clock source, no external crystal is needed to run the chip, and no clock lines are available to the attacker. The entire secure system exists on the die of the chip, making this kind of attack significantly more difficult. By using this internal oscillator, the attack is completely neutralized.

## Brownout Attack

Brownout attacks share many similarities with clock attacks. Both are fault attacks capable of skipping various instructions in the MCU. However, the brownout attack modifies the chip’s power pins, rather than modifying the clock pin. This means that it is impossible to completely neutralize this kind of attack in the same way as the clock attack, as power pins are always required. The attack itself is significantly less reliable, however, and it was found to occasionally damage the chip under test.

To mitigate this attack, multiple steps were taken. First, the ATMega1284P’s internal brownout detection circuit was enabled, and set to the maximum threshold voltage of 4.3 volts. This is a simple setting which places the chip in a reset state if the power pins dip below the threshold voltage for more than 2 microseconds. While effective for large instruction skips, this step alone is not enough, as 2 microseconds still represents a large number of instructions. Therefore, additional steps are taken to obscure knowledge of the program execution state.

The use of an internal RC oscillator is a major help in obscuring the state of program execution, as it denies an attacker access to a clock signal that could be used to keep in time with the instruction counter. The ATMega1284P’s internal clock management hardware is also used to vary the clock speed between the full 7.37 MHz and 1/8 speed, or 0.921 MHz. The speed is changed based on the status of a pseudorandom number generated by a 16-bit maximal-state LFSR (Linear Feedback Shift Register). In addition, random delays inserted in the AES encryption code also make the program execution time non-deterministic. This makes brownout attacks much more difficult, as the attacker is not sure when the desired instruction occurs. While the lower clock speed also contributes to the chip’s resistance to brownout attacks, it was found that glitches would still occur down to 1 MHz.

Originally, it was intended to use the watchdog timer oscillator in conjunction with the system clock as a source of true random numbers in the form of clock jitter. This would allow for the generation of 1 bit of entropy approximately every 16 milliseconds. Testing of the random bits confirmed that the entropy from this source was indeed high. While the code for true random number generation was written, time and program space constraints prevented it from being implemented.

## Power Side Channel Attack

The final kind of attack considered was the power side channel attack. This attack uses information on the power consumption of the ATMega1284P in combination with statistical analysis to determine information about the key used, drastically reducing the possible number of valid keys. As knowledge of the key is not something an attacker should ever have, this attack is very bad.

Mitigating the risk of this attack followed much of the same fashion as mitigating the brownout attack. The changing clock speed and random delays inserted in the AES code lead to a non-deterministic program execution time, making it more difficult for the attacker to sync power traces to the AES encryption and decryption processes. In addition, a side-channel-resistant AES encryption core, based on the avr-crypto-lib implementation was developed. This side-channel resistant code used multiple S-boxes and dummy operations to obscure the AES encryption process. With the current implementation, one would require 96 times as many power traces as normal, due to the 5 dummy operations per byte operation. In addition, all registers used are first initialized to a pseudorandom number, meaning that analyzing the Hamming Distance of values computed would be impossible. Additive masking is used to protect linear operations in AES, meaning the only way to attack the implementation would be to attack the S-box output using a Hamming Weight model.