S-RFUP: Secure Remote Firmware Update Protocol

Rakesh Podder[1(✉)], Tyler Rios[1], Indrajit Ray[1], Presanna Raman[2], and Stefano Righi[2]

[1] Colorado State University, Fort Collins, CO, USA
{rakesh.podder,tyler.rios,indrajit.ray}@colostate.edu
[2] AMI US Holdings Inc., Duluth, GA, USA
{presannar,stefanor}@ami.com

Abstract. Traditional over-the-air (OTA) update mechanisms lack security features. As a result, OTA firmware updates expose a device to several threats including unauthorized update, introduction of malware in the firmware code and rollback of firmware to an vulnerable older version. A handful of domain specific OTA firmware update protocols, especially in the automotive sector, have started incorporating rudimentary security features; however, these are not always enough. Moreover, a lack of standardization can lead to compatibility issues. In this work, we introduce the Secure Remote Firmware Update Protocol (S-RFUP) for platform (We use the term "platform" to mean any computer or hardware device and/or associated operating system, or a virtual environment on which software can be installed and run. Source NISTIR 7698, https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7698.pdf) firmware updates that enhances security and operational integrity across firmware devices during the update procedure. We build upon the hardware root of trust functionality provided by the Project Cerberus to perform secure attestation. With a goal of providing uniformity across a multitude of platforms, we leverage industry standards such as Platform Level Data Model (PLDM), Management Component Transport Protocol (MCTP), and well established cryptographic algorithms. Incorporating PLDM and MCTP reduces the management complexity and ensure interoperability between different hardware and software components in platform. We provide a security analysis of the proposed S-RFUP framework and discuss its implementation, testing and validation results.

Keywords: PLDM · MCTP · Project Cerberus · Firmware Update · Hardware Root of Trust (HRoT)

1 Introduction

Firmware is a critical piece of software in all computing devices, serving as the intermediary between the hardware functionality and the software operations of these devices. Like any other software, firmware needs periodic updates to address vulnerabilities, enhance performance, and introduce new features. Traditionally, firmware updates had been conducted locally. However as devices grow in complexity (think, cloud servers) and scale (think Internet of Things (IoT)) significant challenges arise for local firmware updates, such as physical access requirements, operational downtime, and logistical complexities.

Remote Firmware Update (RFU) is emerging as a viable solution, enabling updates to be deployed over-the-air (OTA) [38], minimizing disruptions and eliminating the need for physical proximity. However, over-the-air (OTA) update mechanisms, while convenient, are often targeted by attackers such as replay attack [39], denial of services [23], legacy firmware update [41], tampering [42], fake and malicious updates [2], and eavesdropping [20]; these attacks allow adversaries to tamper with the firmware, execute arbitrary code or roll back the firmware version to expose prior vulnerabilities [8,37]. Recent vulnerabilities in the update mechanisms of Jeep Cherokee [26], Samsung SmartThings Hub[1], and Asus Router[2] highlight these concerns. Ensuring the security of RFU protects the integrity of the firmware, maintains device functionality, and safeguards sensitive information contained within these devices. This is particularly crucial in industries where compromised firmware could lead to severe operational disruptions.

In this paper, we describe our efforts to achieve high-level security and standardization in firmware updates across platforms. The proposed Secure Remote Firmware Update Protocol (S-RFUP) systematizes the update process and provides needed security features to protect against tampering, unauthorized firmware rollback and intellectual property (IP) theft. S-RFUP ensures compatibility across a diversity of devices, and reduces complexity, which, in turn, also enhances the overall security posture by minimizing the inconsistencies that can be exploited in an attack. S-RFUP leverages Project Cerberus' paradigm of hardware root of trust [21] to enable secure attestation, ensuring that all firmware and software boot processes are verified and secure and making it an ideal foundation for secure RFU. We integrate industry standards such as Platform Level Data Model (PLDM), Management Component Transport Protocol (MCTP) with Project Cerberus for standardizing remote management and monitoring of firmware updates. The proposed S-RFUP framework leverages strong cryptographic such as AES-256 and ECDH Key Exchange techniques to encrypt and decrypt messages during communication between Update Agent and Firmware Device.

The main contributions of this work are:

1. We present a novel framework (S-RFUP) that integrates industry standard protocols (PLDM, MCTP) with a hardware root of trust (HRoT) to ensure interoperable and secure RFU across multiple device platforms.
2. We extend Project Cerberus by introducing new libraries to handle PLDM message construction, PLDM over MCTP binding, and encryption for an end-to-end secure pipeline and standrization of RFU.

[1] CVE-2018-3926: https://nvd.nist.gov/vuln/detail/CVE-2018-3926.
[2] CVE-2021-3166: https://nvd.nist.gov/vuln/detail/CVE-2021-3166.

3. We perform empirical evaluations and security analyses to validate the functionality and effectiveness of the proposed protocol and demonstrate robustness against known vulnerabilities, ensuring a secure firmware update process respectively.

2 Related Works

Security researchers have demonstrated several concerns with conventional remote BIOS and EFI/UEFI firmware updates [1,2,4,20,23,37,39,41,42]. In the following, we discuss some of these works and limitations of proposed solutions.

The integrity of the system BIOS is essential for the security and operational reliability of computer systems, especially for critical systems such as cloud servers, healthcare devices etc. Unauthorized firmware modifications, including malicious firmware updates and alterations, present serious risks [35]. These unauthorized modifications can occur when attackers exploit vulnerabilities or insufficient security practices during a remote firmware update process to manipulate the BIOS, potentially introducing malware or Trojans that compromise data security, disrupt device functionality, or enable system hijacking [3]. The threat landscape is further complicated by man-in-the-middle attacks (MITM) [5] and supply chain attacks [27], which may intercept or tamper with firmware updates or embed malware. Such attacks leverage the foundational level operations of firmware, challenging both detection and mitigation efforts, and pose ongoing security concerns [6,16]. Cui et al. [8] present a proof-of-concept for printer malware that can perform network reconnaissance, extract data, and spread to additional devices. The research points out the inadequacies in the security of firmware update processes and the existence of vulnerabilities within third-party libraries used in firmware.

The widespread adoption of non-secure protocols like HTTP further exposes firmware update processes to potential MITM and backdoor exploits [4,34]. PsycoB0t [24] is a well known example of a worm infection during firmware update of a router. Additionally, there have been cases where vulnerabilities in update procedures are exploited to carry out firmware modification attacks [37]. Jack using "jackpotting" [18] demonstrated that unauthorized firmware modification can be done in ATM machines. Costin showcases the susceptibility of certain Lexmark printers to memory inspection and arbitrary firmware modifications by employing PostScript [7].

Both the academic community and the Internet Engineering Task Force (IETF) are working on creating software update mechanisms for Class 1 and Class 2 devices (Upkit [22,28]) and developing hot-patching techniques such as RapidPatch [17], and Hera [30] that can be potentially used. However, the emphasis in these protocols is on patching low capability devices and not necessarily on the security of the same. There are some works by researchers on designing secure firmware updates [9,19,20,29,36,40]. Falas et al. proposed a Public PUF model [14] for secure firmware updates, but its dependency on high computational resources and complex key management, limits its feasibility for low-power IoT devices and present scalability and standardization challenges in large-scale deployments. The OTA firmware update mechanism proposed by Frisch et al. [15], requires manual intervention to update the framework version and rebuild the firmware, potentially leading to higher latency in updates when API changes occur. Also, the method is unable to verify the cryptographic signature before writing the firmware to flash, potentially allowing corrupted or malicious updates to be partially written before being detected. Similarly, the proposed Narrowband IoT (NB-IoT) as the wireless communication standard for firmware updates by Mahfoudhi et al. [25], does not implement any support for firmware authenticity and confidentiality, and thus can be susceptible to legacy firmware update and MITM attacks. The current state-of-art protocols fails to provide a high-level security with standardization in firmware updates across diverse device ecosystem. Our proposed S-RFUP can provide firmware integrity, and a streamlined and standardized operational functionality to secure sensitive information during remote firmware update.

3 Background

3.1 Firmware Base Specifications

PLDM Base Specification [11] is a standardized protocol designed to facilitate efficient communication and management within platform management subsystems. The primary purpose of the *PLDM Base Specification* is to establish a common protocol for monitoring and controlling various platform components, such as sensors, firmware, and hardware subsystems. This includes inventory management, event notifications, control functions, and data transfer operations.

MCTP Base Specification [10] is a comprehensive communication model designed to facilitate interactions between management controllers and management devices within a platform. MCTP establishes a standardized protocol that can be implemented across various physical transport mediums, enabling flexible and robust platform management solutions. This protocol operates independently of the underlying bus properties and data-link layer messaging. This abstraction allows MCTP to be implemented over different transport bindings such as PCIe, SMBus/I2C, and potentially other mediums like USB and RMII in the future.

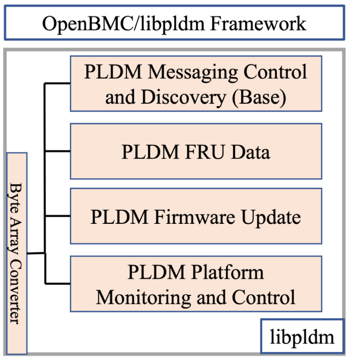
OpenBMC/libpldm Framework
PLDM Messaging Control and Discovery (Base)
PLDM FRU Data
PLDM Firmware Update
PLDM Platform Monitoring and Control
Byte Array Converter
libpldm

Fig. 1. OpenBMC/libpldm library Framework.

PLDM over MCTP Binding Specification [12] outlines how PLDM messages are transported over MCTP, establishing a common format and ensuring interoperability between different hardware and software components within a platform management subsystem. The main objective of the *PLDM over MCTP Binding Specification* is to establish the message format and protocol requirements for transmitting PLDM messages via the MCTP transport protocol.

We are using this specifications in our proposed S-RFUP framework to standardized and streamline the firmware update process.

3.2 PIT-Cerberus

Project Cerberus [21] is developed by Microsoft as a hardware root of trust (HRoT) specifically for server platforms. It enables secure boot functionality for device firmware, whether or not the devices inherently support secure boot. Additionally, it offers a secure method to verify and attest to the firmware state of the devices. We are using Project-Cerberus as a server-platform for Update Agent (UA) and as a HRoT for Firmware Device (FD).

3.3 OpenBMC/libpldm

This library is part of the OpenBMC project, aimed at providing an open-source firmware solution for baseboard management controllers (BMCs). The '*libpldm*' deals with the encoding and decoding of PLDM messages. Figure 1 shows various core module to facilitate tasks such as firmware updates, monitoring, and control of hardware devices across different hardware platforms.

4 Threat Model

In this section, we delineate the threat model pertinent to the secure remote firmware update protocol (S-RFUP).

4.1 Assumptions

For this work we made a number of assumptions. First, we assume the HRoT processor is considered tamper-proof and trusted. The company is deemed trustworthy, with no insider threats, and securely programs the HRoT processor with the PIT-Cerberus and related libraries and data. The Update Agent (UA) is honest and not curious, ensuring the protection of any stored confidential information against breaches of confidentiality and integrity. The company server used by UA is considered a trusted zone, protected by Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), alongside strict security policies and personnel and not a target for intrusion and denial-of-service attacks. Protocols and cryptographic methods, the key size of Advanced Encryption Standard (AES), prime modulus of Elliptic-curve cryptography (ECC), curve selection for ECC, and the Digital Signature Algorithm (DSA), are carefully selected and implemented in a secure environment (SCIF[3]) to resist physical tampering and

[3] Sensitive Compartmented Information Facility.

side-channel attacks. The initial key establishment is completed before deployment and is not regarded as an issue. Encryption keys are generated with strong randomness and installed in the firmware devices during the manufacturing process.

4.2 Attacker Model

The primary target of the attacker is the communication channel between UA, and FD during firmware updates. We adopt the Dolev-Yao attacker model [13], where the attacker can eavesdrop, intercept, modify, or inject messages into the communication channel. Replay attacks involve attackers delaying or re-sending packets to mislead the FD or UA. Attackers might also inject false information to disrupt ongoing services. An attacker impersonating (MITM attacks) the UA or a legitimate FD could compromise firmware integrity or gain unauthorized access. The FD is vulnerable to physical tampering during transit, such as replacing the HRoT with a malicious microcontroller or embedding a hardware Trojan. To mitigate this, tamper-proof seals are assumed to protect the device during shipment, alerting the Recipient if breached. This paper does not consider scenarios involving physical tampering with the Firmware Device.

4.3 Desired Security Properties

The essential security properties required for network traffic within a secure remote firmware update protocol include data integrity, data authentication, data confidentiality, and data freshness.

- **Data Integrity**: Ensures that the firmware updates received by the Firmware Device (FD) have not been tampered with.
- **Data Authentication**: Verifies the source and integrity of the firmware update to ensure that the updates come from legitimate sources and prevent unauthorized modifications.
- **Data Confidentiality**: Maintains the secrecy of firmware data by protecting sensitive proprietary information from unauthorized access.
- **Data Freshness**: Guarantees that the firmware updates are recent.

By addressing these security properties, the protocol (S-RFUP) aims to provide a robust, secure, and reliable process for remote firmware updates.

5 Description of S-RFUP

The S-RFUP operates as a client-server model, as depicted in Fig. 2. The key entities within this protocol include the Hardware Root of Trust (HRoT), Firmware Devices (FD), Update Agent (UA), Sender, and Recipient. UA is a function within S-RFUP framework, designed to identify firmware devices, capable of executing a PLDM firmware update and to facilitate the transfer of component images to these devices. HRoT is the tamper-proof micro-controller

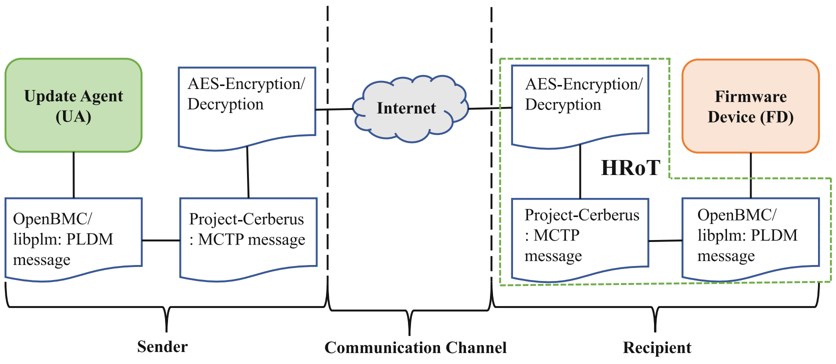
Update Agent (UA)
AES-Encryption/ Decryption
Internet
AES-Encryption/ Decryption
Firmware Device (FD)
HRoT
OpenBMC/ libplm: PLDM message
Project-Cerberus : MCTP message
Project-Cerberus : MCTP message
OpenBMC/ libplm: PLDM message
Sender
Communication Channel
Recipient

Fig. 2. High-level representation of S-RFUP framework.

that acts as the hardware root of trust. It is a critical component leveraging the Project Cerberus embedded framework for Firmware Device. FD is a PLDM endpoint (terminus) that comprises one or more processor elements that execute firmware. The Sender is the manufacturer or company responsible for maintaining the firmware and initiating firmware updates, while the Recipient is the end user utilizing the firmware device with HRoT capabilities.

5.1 Proposed Approach

The proposed S-RFUP framework is divided into two main segments; 1) the establishment of a secure channel, 2) the initiation of remote firmware update process. Initially, UA and FD establish a connection and compute public and private key pairs using Elliptic-curve Diffie-Hellman (ECDH) key agreement protocol [31,33]. UA generates an ECC (Elliptic Curve Cryptography) private key (d_U) - public key (q_U) pair, where, $q_U = d_U \times G$. G is the base point of the chosen elliptic curve. Then UA sends the public key (q_U) to the FD. FD generates a private key d_F, computes a public key $q_F = d_F \times G$ and an AES secret key $S = d_F \times q_U$. FD transmits its public key to the UA, which then computes the same AES secret key S as $S = d_U \times q_F = d_U \times \{d_F \times G\} = d_F \times \{d_U \times G\} = d_F \times q_U$.

In the second segment, as illustrated in Fig. 3, UA initiates the firmware updates, converting the firmware image into a Platform Level Data Model payload (*pldm* message) using '*OpenBMC/libplm*' library. The *pldm* message is then transformed to a *mctp* message by the Project-Cerberus using Management Component Transport Protocol (MCTP) & S-RFUP. After that, the *mctp* message is encrypted with AES encryption [32] schema ($encData = AESEncryption(mctp, S)$) using a shared key ($S$) generated previously by UA, before sending it to HRoT. HRoT, containing the Project Cerberus framework (MCTP Protocol) with S-RFUP functionalities, decrypts it using AES ($mctp = AESEncryption(encData, S)$) and converts the *mctp* to *pldm* message, before sending it to FD. Based on the request data (*pldm* message) FD generates an response data using '*libpldm*' and sends it back to UA.

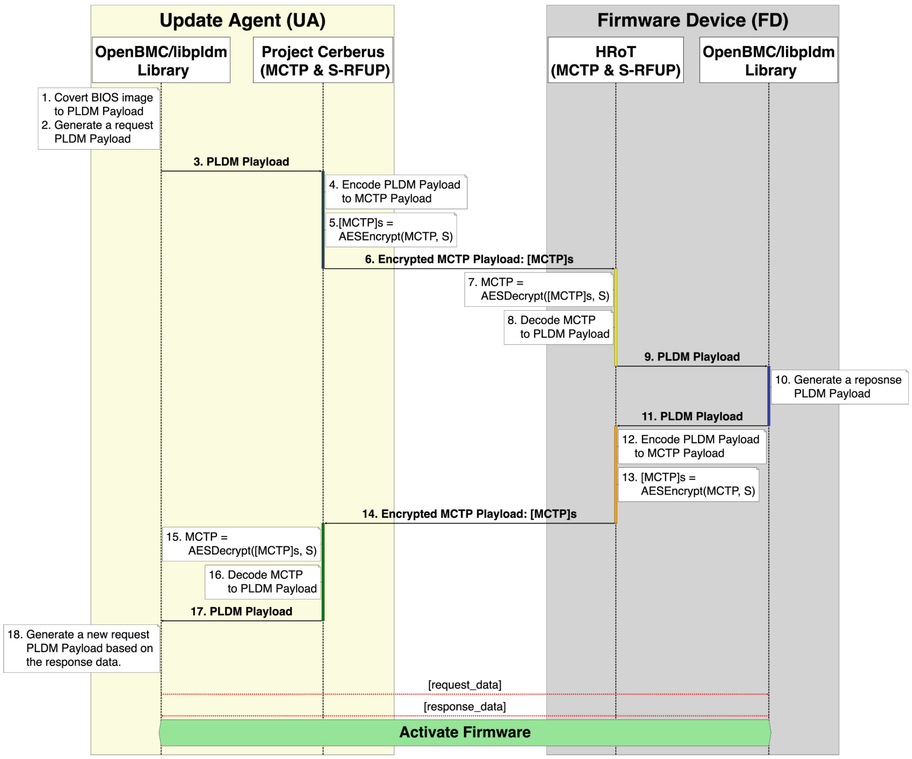
Update Agent (UA)
OpenBMC/libpldm Library
Project Cerberus (MCTP & S-RFUP)
Firmware Device (FD)
HRoT (MCTP & S-RFUP)
OpenBMC/libpldm Library
1. Covert BIOS image to PLDM Payload
2. Generate a request PLDM Payload
3. PLDM Playload
4. Encode PLDM Payload to MCTP Payload
5.[MCTP]s = AESEncrypt(MCTP, S)
6. Encrypted MCTP Playload: [MCTP]s
7. MCTP = AESDecrypt([MCTP]s, S)
8. Decode MCTP to PLDM Payload
9. PLDM Playload
10. Generate a reposnse PLDM Payload
11. PLDM Playload
12. Encode PLDM Payload to MCTP Payload
13. [MCTP]s = AESEncrypt(MCTP, S)
14. Encrypted MCTP Playload: [MCTP]s
15. MCTP = AESDecrypt([MCTP]s, S)
16. Decode MCTP to PLDM Payload
17. PLDM Playload
18. Generate a new request PLDM Payload based on the response data.
[request_data]
[response_data]
Activate Firmware

Fig. 3. Sequence diagram of S-RFUP firmware update process.

Similarly, if FD has request data, it follows the same encoding, encrypting, transferring, decrypting, and decoding steps, ultimately returning the response data to UA. This process will continue till all the firmware components (including firmware package header and payload) transfer to FD and FD activates the firmware updates.

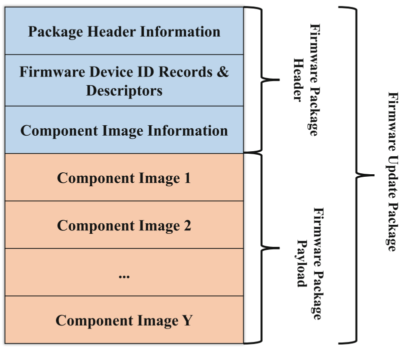

Fig. 4. PLDM firmware update package.

5.2 PLDM Firmware Update Package

The firmware update package is designed to work in conjunction with PLDM Firmware Update commands and contains several essential elements. These elements include a firmware package header that outlines the update package's contents as illustrated in Fig. 4. Specifically, the header provides a description of the overall packaging version and the date it was created. It also includes device identifier records, which specify the firmware devices (FDs) targeted for the update. Further, the header details the package contents, listing each component image's classification, offset, size, and version. Additionally, the package incorporates a checksum to ensure the integrity of the data.

Within the S-RFUP framework, three primary types of PLDM commands facilitate the transfer of firmware package headers and component images during updates. These include: {Inventory: QueryDeviceIdentifiers, GetFirmwareParameters},{Update: RequestUpdate, PassComponentTable, UpdateComponent, TransferComplete, VerifyComplete, ApplyComplete, ActivateFirmware, GetStatus, CancelUpdateComponent, CancelUpdate}, and {Transfer: RequestFirmwareData, GetPackageData, GetDeviceMetaData, GetMetaData}. These commands utilize the '*OpenBMC/libpldm*' library for efficient serialization and de-serialization of the PLDM messages.

6 Implementation of S-RFUP

In this section we discussed the implementation and execution flow of S-RFUP within the Project Cerberus framework. It outlines the libraries, procedural steps, and interactions between various components involved in the firmware update process. Figure 5 illustrates a generic *mctp* message that has encapsulated a *pldm* message. This *pldm* message is generated by UA using '*libpldm*' library. For each PLDM command described in Sect. 5.2, the fields of *pldm* message will be populated with different values. Once the *pldm* message is generated it will be encoded to *mctp* message shown in Fig. 5 before encrypting or decrypting. For this purpose, we developed the following core libraries that could handle the firmware update process (Table 1).

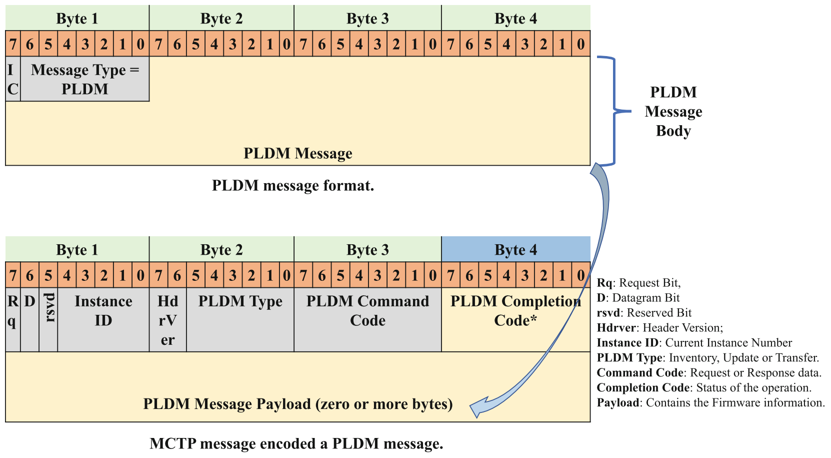

Fig. 5. Generic MCTP message encoded a generic PLDM message.

6.1 S-RFUP Core Libraries

Table 1. Description of S-RFUP core libraries.

Source	**Function/API**
New S-RFUP Libraries	
pldm_fwup_crypto	`keyGeneration(), keyExachange(), secretKey() AESEncryption(), AESDecryption(), generateDSA()`
cmd_interface_pldm	`cmd_interface_pldm_process_request() ,cmd_interface_pldm_process_response()`
pldm_fwup_handler	`pldm_fwup_handler_run_update_ua(), pldm_fwup_handler_start_update_fd()`
pldm_fwup_manager	`pldm_fwup_manager_init(), pldm_fwup_manager_deinit().`
pldm_fwup_protocol_commands	`pldm_fwup_process_query_device_identifiers_request(), pldm_fwup_prcocess_get_firmware_parameters_request(), pldm_fwup_process_request_update_request(), pldm_fwup_process_request_update_response(), ..., pldm_fwup_generate_activate_firmware_request(), pldm_fwup_generate_activate_firmware_response()`
pldm_fwup_protocol	`struct pldm_fwup_protocol_version_string, struct pldm_fwup_fup_component_image_entry, struct pldm_fwup_protocol_component_parameter_entry`
Modified Project Cerberus Libraries	
core/mctp	`mctp_interface_process_packet()`
core/projects/linux	`platform_config()`
core/tools/testing	`setup_fwup_flash_virtual_disk()`

The S-RFUP architecture employs a modular approach to firmware updates. The core modules of S-RFUP and their interaction with various external framework such as Project Cerberus and '*OpenBMC/libplm*', is illustrated in Fig. 6.

PLDM FWUP Crypto. The '*pldm_fwup_crypto*' supports the encryption and decryption of messages, securing communications across the network. It utilised the `aes.h` & `ecc.h` to generate ECDH key pair and AES encryption from Project Cerberus. A *Lamport* timestamp is incorporated within `AESEncryption()` & `AESDecryption()` before encrypting/decrypting the *mctp* message. We are also using ECC curve to generate a digital signature (DSA) to sign each encrypted *mctp* message for UA and FD. The `signature_verification_ecc.h` file helps to verify the figital signatures of UA/FD during update process.

PLDM Command Interface. Project Cerberus (or Cerberus) defines a generic command interface called '*cmd_interface*' for processing requests and responses in a command protocol. The '*cmd_interface_pldm*' extends '*cmd_interface*' to handle PLDM specific commands as shown in Table 1. It inherits the properties and function pointers from '*cmd_interface*' which are then defined during its initialization. Currently '*cmd_interface_ pldm*' only processes PLDM firmware update command types, but can be further extended to process others such as PLDM for FRU commands.

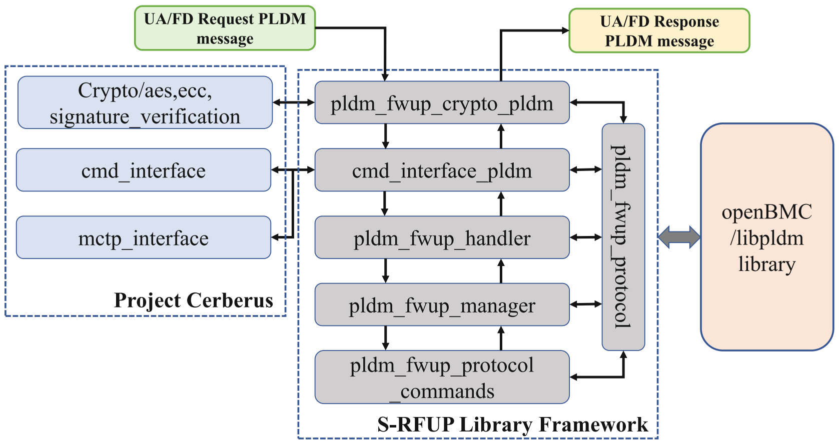

Fig. 6. S-RFUP library framework and interaction with Project Cerberus & *libpldm*.

Command Interfaces and MCTP: Cerberus uses MCTP as the protocol for which messages are exchanged throughout a Cerberus managed subsystem. MCTP is a flexible standard that can encapsulate other protocols such as Cerberus's own command protocol, SPDM. We modified the `mctp_interface_process_packet()` function to handle PLDM command. During the processing of MCTP packets Cerberus will interpret the MCTP header and extract the message type field which descries the type of payload that packet is carrying. The payload is then passed along to its respective command interface for further processing.

PLDM FWUP Manager. The '*pldm_fwup_manager*' is a library for managing the state of a PLDM-based firmware update and allowing other parts of the S-RFUP framework to modify or view the information present in the firmware update commands. An instance of it is passed along to the '*cmd_interface_pldm*' so that during the processing of firmware update commands the information needed to populate or save the fields of the commands can be accomplished.

PLDM FWUP Protocol Commands. The '*pldm_fwup_protocol_commands*' contains functions which perform the actual decoding and encoding of PLDM commands saving information to or populating message fields with information from the PLDM FWUP manager. For example, '`pldm_fwup_process_request_update_request()`' function is used to process incoming RequestUpdate PLDM commands saving information in the request data to the manager and extracting the manager's context to generate a RequestUpdate response.

PLDM FWUP Handler. The '*pldm_fwup_handler*' is the main driver code of S-RFUP. The handler mainly calls the API of the PLDM FWUP protocol commands and the MCTP interface API to generate, send, receive, process, and respond to PLDM firmware update commands. The most important fact is that at any time S-RFUP can be operating as either UA, performing firmware update on another device it manages or as the actual FD being updated. As such the '*pldm_fwup_handler*' interface contains two function pointers: `pldm_fwup_handler _run_update_ua()` for updating a firmware device in the subsystem as an UA and `pldm_fwup_handler_run_update_fd()` for updating S-RFUP's own firmware (FD) as directed by another UA.

Apart form this core libraries, S-RFUP also has '*pldm_fwup_protocol*' library header containing various structures, macros, and enumerations used by the above mentioned libraries. Table 1 shows the main libraries and some of API that we have designed, modified or used in the protocol. To convert the BIOS/BMC firmware images to a PLDM message we are mainly using `firmware_update.h` of '*libpldm*' library as external.

As explained in Sect. 5.2, the firmware component images reside in the Firmware Update Package where they are retrieved as needed using Cerberus's flash module. Since we are compiling and evaluating on Linux, a virtual flash module was created to simulate that functionality using disk I/O. We have developed a python script `setup_fwup _flash_virtual_disk()`, that located in the '*core/tools*' directory, generates a 4GB binary file divided into sections to simulate different flash regions: one for package data, one for meta data, and one for each two firmware components. These regions are populated with random bytes.

6.2 State Transitions

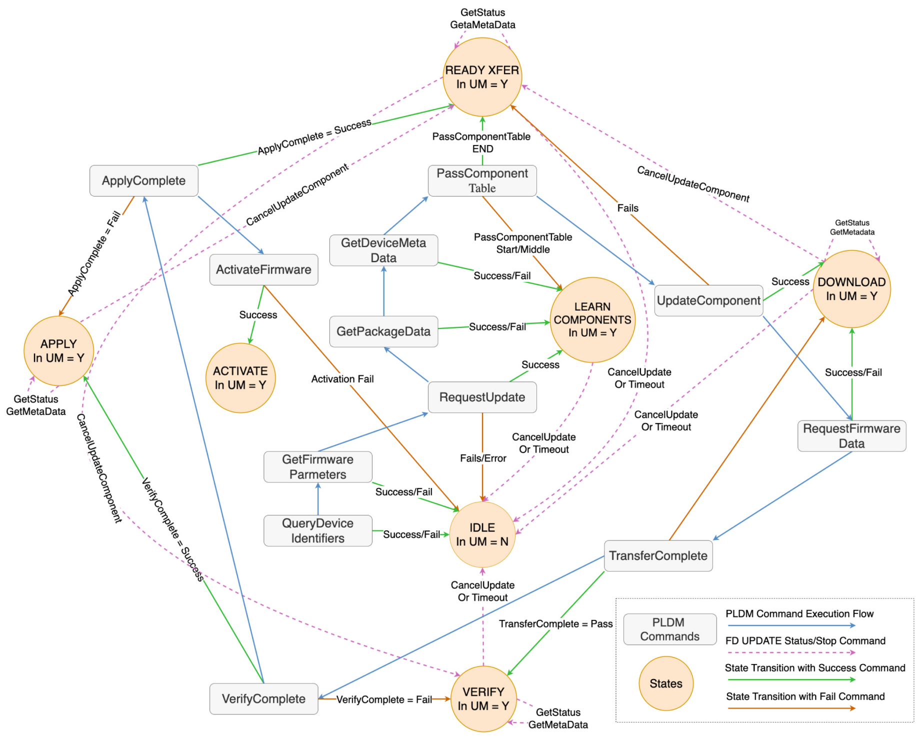

Fig. 7. State Transition Diagram with PLDM Commands.

Figure 7 outlines the various PLDM command and states that the Firmware Device (FD) can be during update process. Each circle in the diagram represents a distinct state the FD might be in and, each rectangular boxes represents a PLDM command. Whenever the FD is initialized, or when it undergoes a system reboot or device reset, it starts in the IDLE state. The execution starts from QueryDeviceIdentifiers and ends at ActivateFirmware. Every PLDM command on a successful execution transits to next PLDM command and the associated states also change. For example, if S-RFUP executes RequestUpdate command, on success it will move to GetPackageData and state of FD will change from IDLE to LEARN COMPONENTS. Similarly, if the execution fails or FD throws a compilation code error, the state of FD remains on IDLE. This design helps us to understand the FD's state with each PLDM command, so that a prompt diagnosis can be launched if any error occurs and it also helps to standardise the update process.

6.3 S-RFUP Update Flow in Project-Cerberus

The firmware update is performed in a sequential manner. Implementation for parallel operation and message exchange is not addressed in the paper. Additionally, the '*pldm_fwup_manager*' must be initialized and the PLDM command interface must obtain a reference to the manager prior to the start of the firmware update process.

Before UA initiates the update process, `keyGeneration()` function of '*pldm_fwup _crypto*' library is responsible for generating the ECC key pair for UA and FD. It loads the length of the key in *key_length*, initializes private key in *privkey*, public key in *pubkey*. `keyExchange()`: function exchanges the public key of UA and FD. On success, `keyExchange()` will initialize *pubkey_cli* with the UA's public key and load the *pubkey_serv* variable with a public key received from the FD. The `secretKey()` takes ECC private key from UA and FD, computes the secret key and loads in *secret* parameter.

S-RFUP Operating as UA Flow

1. First, S-RFUP checks the need to send QueryDeviceIdentifiers and GetFirmwareParameters based on the control boolean in `pldm_fwup_handler_run_update_ua()`.
2. It then sends these inventory commands followed by the RequestUpdate command, passing information such as maximum transfer size and number of allowed outstanding requests, configurable in `platform_config()`.
3. If package data is required, the device notifies S-RFUP, which then sends the necessary data via GetPackageData. Subsequently, S-RFUP may also send the GetDeviceMetaData command depending on the device's feedback.
4. Next, S-RFUP sends the PassComponentTable command, transferring component details to the device, which responds with compatibility codes.
5. Upon confirming compatibility, S-RFUP issues the UpdateComponent command for each component sequentially.

6. As the device requests firmware data using the RequestFirmwareData command, S-RFUP responds with the specified firmware portions.
7. After the firmware transfer, the device sends a TransferComplete command. If there is an error, S-RFUP might send a CancelUpdateComponent command.
8. S-RFUP waits for the VerifyComplete command within a preset timeout period, monitoring the status with potential GetStatus commands.
9. Once verification is complete, the device is expected to apply the update, followed by an ApplyComplete command.
10. These steps are repeated for each component until all are updated.
11. Finally, S-RFUP sends ActivateFirmware to activate the firmware, indicating with a boolean flag whether self-contained components should be activated immediately.

Please note that before sending the mctp message `AESEncryption()` encrypts the message using a secret key gnerated by `secretKey()`. This function takes secret key, message plus a timestamp, and use AES-GCM-256 method to encrypt the message and loads into *ciphertext* parameter. Each message then signed with a digital signature generate by `generateDSA()` function. Once the FD receives the message it verifies the signature. `AESDecryption()` function takes encrypted message (*ciphertext*), secret key (*secret*) as input, decrypts it and loads the message to the provided *plaintext* buffer. In the current implementation this metadata is written to a region in flash although depending on the metadat, it could be written to a structure or any other volatile memory.

S-RFUP Operating as FD Flow

1. S-RFUP waits for initial commands from the Update Agent (UA). If an inventory command is received, it anticipates a second inventory command.
2. Upon receiving inventory commands, S-RFUP processes the RequestUpdate command. It determines whether to send the GetPackageData command based on the UA's instructions and specifies the length of metadata to retain.
3. Following the RequestUpdate, S-RFUP issues the GetPackageData and handles any GetDeviceMetaData commands as required.
4. S-RFUP waits to receive the PassComponentTable command from the UA, which specifies which firmware components are to be updated.
5. Upon receiving the UpdateComponent command, S-RFUP verifies component compatibility and proceeds to request the necessary firmware data using the RequestFirmwareData command, specifying the needed data offset and length.
6. After receiving and writing firmware data to flash memory, S-RFUP issues a TransferComplete command, assuming successful transfer unless indicated otherwise.
7. S-RFUP verifies the firmware image by issuing a GetMetaData command, and comparing the received digital signature against its own.
8. Once verification is complete, S-RFUP directly applies the firmware image, and sends an ApplyComplete command to the UA.

9. Steps 5 to 8 are repeated for each firmware component requiring an update.
10. Finally, upon completing all updates, S-RFUP awaits the ActivateFirmware command to activate the updated firmware, with specifics such as timing handled according to settings in `platform_config()`.

The verification mechanism is left up to the user to implement and subsequently the assignment of the result field in the VerifyComplete command (by default set to success). In our case, S-RFUP issues the GetMetaData command with the UA responding with signatures[4] of the firmware image. S-RFUP would then take these and compare them to the DSA signature it generated with the received component image. Additionally, how the firmware is activated depends on the system specifications and user preferences.

6.4 Results and Discussions

We developed various experimental test scenarios to evaluate our framework for correctness, consistency and performance. We run our experiments on 2 virtual Linux servers. The client side (assumed as FD) has a 5000 MHz 12th Gen Intel(R) Core(TM) i7-12700K processor, x86_64 architecture, 20 cpus and UA (as server) in Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60 GHz, x86_64 architecture, 12 CPU(s).

All the S-FRUP PLDM firmware update libraries are available on GitHub[5]. All user guidelines, API descriptions, test results, and setup manuals are publicly available on GitHub.

In order to validate that the our protocol is working as it is supposed to, we used Project Cerberus with S-RFUP libraries (server platform) and '*OpenBMC/libpldm*' library as an Update Agent (UA) or server and a HRoT that is a micro-controller containing Project Cerberus with S-RFUP libraries and '*OpenBMC/libpldm*' library as a Firmware Device or client. We introduce '*cmd_chanel_tcp*' that uses a TCP socket for communication between UA and FD. It has `initialize_global_server_socket()` function that works with '*pldm_fwup_crypto*' & '*cmd_interface_pldm*' library to send the encrypted *mctp* packets to UA/FD. As designed and expected, our protocol delivers the anticipated results, with successful operations observed across both UA and FD.

[4] DSA with keys derived from ECC curve25519 has been used to generate signatures.
[5] https://github.com/AMIProject0/Project-Cerberus-PLDM/tree/master.

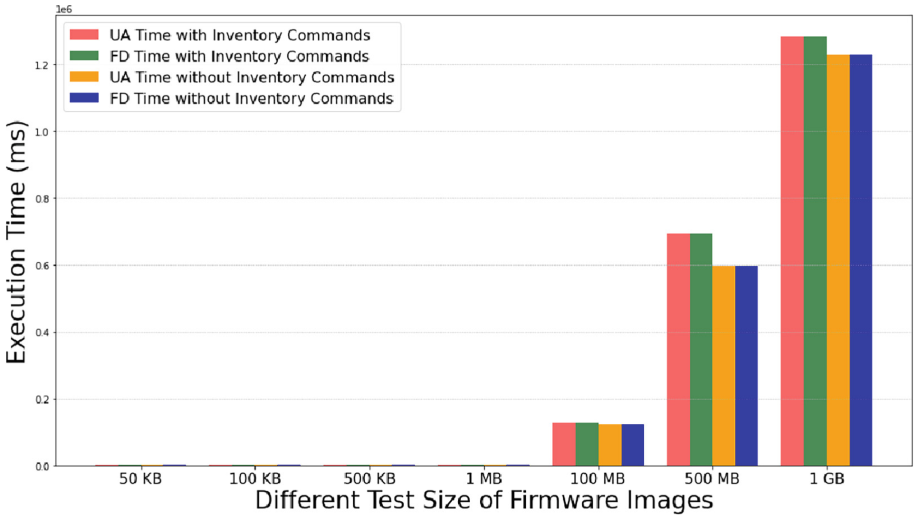

Fig. 8. UA Time vs FD Time for Firmware Update Tests (with and without Inventory Commands).

We have tested our framework with various firmware image sizes:{ 50 KB, 100 KB, . . . , 500 MB, 1 GB }. Figure 8 illustrates a bar chart comparing the execution times for different firmware image sizes, both with and without inventory commands (as it is not always required for inventory commands). For 50 KB to 1 MB, the percentage increases in times are 55.68% (for UA) and 55.56% (for FD). If we compare the 50 KB to 1 GB sizes, the percentage increases in times are 60194.30% (UA) and 60190.13% (FD). The growth rate is exponential. But it is reasonable as, for 1 GB file it's takes only 128 s which is acceptable compare to state-of-art firmware update methods.

6.5 Exception and Error Handling

We have done an extensive software testing for the proposed protocol (S-RFUP). Throughout the development we have introduced several methods to handle errors of the compilation code, time-out exception for each PLDM command and tested the software in various test scenarios that shows the capability of the proposed framework.

Error Completion Codes. For each command, we have designed a specific structure to handle the response based on the compilation codes returned by a *pldm* message from UA/FD. Table 2 shows some of the PLDM commands and various scenarios of compilation code and they to handle the error. Let's say if UA send a RequestUpdate *pldm* message to FD and FD is currently on another firmware update process, the response *pldm* would contain a compilation_code =

ALREADY_IN_UPDATE_MODE and a return value of 0×81. If the FD can not do a firmware update right now it will send a response with compilation_code = *RETRY_REQUEST_UPDATE*. Similarly, we have designed error handling capabilities of each command for all possible states of FD.

Table 2. Command Responses and Descriptions for Firmware Updates. PBC: PLDM_BASE_CODES; AIUM: ALREADY_IN_UPDATE_MODE; UTIU: UNABLE_TO_INITIATE_UPDATE; RRU: RETRY_REQUEST_UPDATE; ITL: INVALID_TRANSFER_LENGTH; CNE: COMMAND_NOT_EXPECTED; DOFR: DATA_OUT_OF_RANGE; RRFD: RETRY_REQUEST_FW_DATA; CP: CANCEL_PENDING;

Command Name	Completion Codes	Return Value	Return By	Descriptions
QDI	PBC	0x00	FD	Executed successfully.
RU	PBC	0x00	FD	Executed successfully.
	AIUM	0x81		Already in update mode.
	UTIU	0x8A		Unable to enter update mode.
	RRU	0x8E		Requests a retry of the RequestUpdate command, needing more time to prepare.
RFD	PBC	0x00	UA	Executed successfully
	ITL	0x83		Image portion > MaxTransferSize .
	CNE	0x88		Command is not expected in the sequence.
	DOFR	0x89		Image portion offset exceeds the range.
	RRFD	0x91		Component image portion is not available.
	CP	0x87		When CancelUpdate initiated by FD previously.

Timing Specification A timing specification has been designed for every compilation codes and time-out exceptions. For RequestUpdate response message if compilation_code = *RETRY_REQUEST_UPDATE* sent by FD, it will assign an UA_T4[6] time specification for the process. It means the amount of time to wait before UA re-sends a RequestUpdate PLDM command after receiving the previous response. There are also, GetPackageData timeout ($1s \leq$ UA_T5 $\leq 5s$), Update mode IDLE timeout for FD ($60s \leq$ FD_T1 $\leq 120s$) and several others that we have specified. A detailed documentation about the timing specifications will be provided with the source code.

We have also designed several test scenarios to check various failure that can occur during firmware update process such as, if the UA/FD loses connections during update process then the program will wait until the timer specified time (UA_T7 or FD_T5) or if the connection is not back, it will revert back to the previous IDLE state and throws a timeout exception (GT_T1). The S-RFUP is thoroughly tested and validated to handle unexpected behaviours during firmware update process.

[6] For UA_T4 the $minTime = 1s$ and $MaxTime = 5s$.

7 Security Analysis

The proposed secure firmware update protocol (S-RFUP) effectively mitigates a range of security threats through a combination of robust encryption techniques, rigorous verification procedures, and systematic error handling mechanisms. The following analysis details how the identified threats are addressed using this approach and how the essential security properties are maintained. To prevent service unavailability (due to dos attack) and ensure continuous operation during the firmware update process, the protocol incorporates several exceptions and error handling mechanisms. These mechanisms are designed to catch and manage unexpected errors or attacks that could lead to server or service unavailability. By handling such scenarios promptly, the system avoids unwanted interruptions or crashes, maintaining service availability and reliability.

The communication between the UA and FD is protected using advanced encryption methods. Specifically, the Management Component Transport Protocol (MCTP) messages are encrypted using the S-RFUP 'crypto' library framework, which utilizes the Advanced Encryption Standard – Galois/Counter Mode – with a 256-bit key (AES-GCM 256). This encryption method is resistant to a variety of attacks, including *Known Plaintext Attack, (KPA)*, *Chosen Plaintext Attack (CPA)*, *Chosen Ciphertext Attack (CCA)*, and *Ciphertext-Only Attack (COA)*. By encrypting all communications, the protocol ensures that any intercepted data remains inaccessible to attackers, protecting sensitive information such as firmware/device information. Thus, data integrity is maintained by ensuring as data packets are delivered to the Recipient without any alterations. The protocol achieves this through the use of AES-GCM encryption, which includes built-in integrity checks to verify that the data has not been tampered with during transmission.

To mitigate replay attacks, the protocol employs unique session keys and *Lamport* timestamps. During each session, the UA and FD generate a new shared AES-GCM key using the ECDH key agreement protocol. This ensures that each session is encrypted with a unique key, preventing attackers from reusing intercepted messages in a different session. Thus, by employing *Lamport* timestamps we can verify the freshness of messages, ensuring that old messages cannot be replayed to disrupt the firmware update process, this gives the assurance of data freshness.

The protocol incorporates the Digital Signature Algorithm (DSA) with keys derived from ECC curve to authenticate legitimacy of the entities. This ensures that the recipient can verify the authenticity of the sender, protecting against impersonation attacks. By using digital signatures, the protocol ensures that only legitimate UA and FD entities can participate in the firmware update process, effectively preventing MITM attacks where an attacker could intercept and alter communications. This process confirms the authenticity of the data source and the integrity of the data itself. The use of ECDH for key exchange, combined with AES-GCM for encryption, provides robust protection against *Adaptive Chosen-Plaintext* and *Chosen-Ciphertext Attacks*. By generating a new secret key for each session, the protocol ensures that attackers cannot use previous encryption or decryption to infer the current encryption key. This dynamic key management system enhances security by preventing key reuse and complicating any attempts to compromise the communication through adaptive attacks.

This comprehensive security framework provides robust protection against eavesdropping, replay & MitM attacks, credential theft, and adaptive attacks, ensuring a secure and reliable remote firmware update mechanism. S-RFUP successfully maintains the essential security properties of data integrity, data authentication, data confidentiality, and data freshness, providing a secure, and reliable process for remote updates.

8 Conclusion

In this work, we propose S-RFUP as a uniform framework for secure remote firmware updates across mutitude of platforms. S-RFUP builds upon Project Cerberus hardware root of trust capabilities by integrating the industry-standard protocols PLDM & MCTP and conventional cryptographic protocols to ensure a secure, reliable, interoperable and easily manageable firmware update process. The implementation has been rigorously tested, validating its resilience against various security concerns and demonstrating its robustness in a controlled environment. We plan to open-source S-RFUP libraries. Future work involves enhancing the protocol's performance and security through implementing parallel firmware updates to increase efficiency, porting the hardware-agnostic S-RFUP to the microchip-specific I2C protocol, and validating the protocol across different firmware ecosystems to ensure robust performance and compatibility.

Acknowledgments. This work was partially supported by the U.S. National Science Foundation under Grant No. 1822118 and 2226232, the member partners of the NSF IUCRC Center for Cyber Security Analytics and Automation - AMI, NewPush, Cyber Risk Research, NIST and ARL - the State of Colorado (grant #SB 18-086) and the authors' institutions. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, or other organizations.

References

1. Alrawi, O., Lever, C., Antonakakis, M., Monrose, F.: SOK: security evaluation of home-based iot deployments. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1362–1380. IEEE (2019)
2. Basnight, Z., Butts, J., Lopez, J., Jr., Dube, T.: Firmware modification attacks on programmable logic controllers. Int. J. Crit. Infrastruct. Prot. **6**(2), 76–84 (2013)
3. Basnight, Z., Butts, J., Lopez, J., Jr., Dube, T.: Firmware modification attacks on programmable logic controllers. Int. J. Crit. Infrastruct. Prot. **6**(2), 76–84 (2013). https://doi.org/10.1016/j.ijcip.2013.04.004
4. Bellissimo, A., Burgess, J., Fu, K.: Secure software updates: disappointments and new challenges. In: HotSec (2006)
5. Conti, M., Dragoni, N., Lesyk, V.: A survey of man in the middle attacks. IEEE Commun. Surv. Tutorials **18**(3), 2027–2051 (2016)

6. Cooper, D., Polk, W., Regenscheid, A., Souppaya, M., et al.: Bios Protection Guidelines, vol. 800, p. 147. NIST Special Publication (2011)
7. Costin, A.: Hacking MFPS. In: The 28th Chaos Communication Congress (2011)
8. Cui, A., Costello, M., Stolfo, S.: When firmware modifications attack: a case study of embedded exploitation. In: NDSS (2013)
9. Dhakal, S., Jaafar, F., Zavarsky, P.: Private blockchain network for IoT device firmware integrity verification and update. In: 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), pp. 164–170. IEEE (2019)
10. DMTF: Mctp base specification 1.2.0. DSP0236 (2009). http://dmtf.org/sites/default/files/standards/documents/DSP0236_1.2.0.pdf
11. DMTF: Platform level data model (pldm) base specification 1.0. DSP0240 (2009). http://dmtf.org/sites/default/files/standards/documents/DSP0240_1.0.0.pdf
12. DMTF: Platform level data model (pldm) for firmware update specification 1.0.1. DSP0267 (2009). https://dmtf.org/sites/default/files/standards/documents/DSP0267_1.0.1.pdf
13. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
14. Falas, S., Konstantinou, C., Michael, M.K.: A modular end-to-end framework for secure firmware updates on embedded systems. ACM J. Emerg. Technol. Comput. Syst. (JETC) **18**(1), 1–19 (2021)
15. Frisch, D., Reißmann, S., Pape, C.: An over the air update mechanism for esp8266 microcontrollers. In: Proceedings of the ICSNC, the Twelfth International Conference on Systems and Networks Communications, Athens, Greece, pp. 8–12 (2017)
16. Fuchs, A., Krauß, C., Repp, J.: Advanced remote firmware upgrades using TPM 2.0. In: Hoepman, J.-H., Katzenbeisser, S. (eds.) SEC 2016. IAICT, vol. 471, pp. 276–289. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33630-5_19
17. He, Y., et al.: {RapidPatch}: firmware hotpatching for {Real-Time} embedded devices. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 2225–2242 (2022)
18. Jack, B.: Jackpotting automated teller machines redux. Black Hat USA (2010)
19. Jain, N., Mali, S.G., Kulkarni, S.: Infield firmware update: challenges and solutions. In: 2016 International Conference on Communication and Signal Processing (ICCSP), pp. 1232–1236. IEEE (2016)
20. Keleman, L., Matić, D., Popović, M., Kaštelan, I.: Secure firmware update in embedded systems. In: 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), pp. 16–19. IEEE (2019)
21. Kelly, B.: Project cerberus security architecture overview specification. Open Compute Project (2017). https://learn.microsoft.com/en-us/azure/security/fundamentals/project-cerberus
22. Langiu, A., Boano, C.A., Schuß, M., Römer, K.: Upkit: an open-source, portable, and lightweight update framework for constrained IoT devices. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 2101–2112. IEEE (2019)
23. Lau, P.T., Katzenbeisser, S.: Firmware-based dos attacks in wireless sensor network. In: Katsikas, S., et al. (eds.) ESORICS 2023. LNCS, vol. 14399, pp. 214–232. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-54129-2_13
24. Maassen, A.: Network bluepill-stealth router-based botnet has been ddosing dronebl for the last couple of weeks (2009). https://www.dronebl.org/blog/8
25. Mahfoudhi, F., Sultania, A.K., Famaey, J.: Over-the-air firmware updates for constrained NB-IoT devices. Sensors **22**(19), 7572 (2022)

26. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle. Black Hat USA **2015**(S 91), 1–91 (2015)
27. Miller, J.F.: Supply chain attack framework and attack patterns. The MITRE Corporation, MacLean, VA (2013)
28. Moran, B., Tschofenig, H., Brown, D., Meriac, M.: A firmware update architecture for internet of things. Internet Requests for Comments, RFC Editor, RFC **9019** (2021)
29. Neves, B.P., Santos, V.D., Valente, A.: Innovative firmware update method to microcontrollers during runtime. Electronics **13**(7), 1328 (2024)
30. Niesler, C., Surminski, S., Davi, L.: Hera: Hotpatching of embedded real-time applications. In: NDSS (2021)
31. Podder, R., Abdelgawad, M., Ray, I., Ray, I., Santharam, M., Righi, S.: Correctness and security analysis of the protection in transit (pit) protocol. Available at SSRN 4980331 (2024)
32. Podder, R., Barai, R.K.: Hybrid encryption algorithm for the data security of esp32 based IoT-enabled robots. In: 2021 Innovations in Energy Management and Renewable Resources (52042), pp. 1–5. IEEE (2021)
33. Podder, R., Sovereign, J., Ray, I., Santharam, M.B., Righi, S.: The pit-cerberus framework: preventing device tampering during transit. In: 2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS), pp. 584–595. IEEE (2024)
34. Samuel, J., Mathewson, N., Cappos, J., Dingledine, R.: Survivable key compromise in software update systems. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 61–72 (2010)
35. Schmidt, S., Tausig, M., Hudler, M., Simhandl, G.: Secure firmware update over the air in the internet of things focusing on flexibility and feasibility. In: Internet of Things Software Update Workshop (IoTSU). Proceeding (2016)
36. Sun, S.: Design and implementation of partial firmware upgrade (2019)
37. Tsang, R., et al.: Fandemic: firmware attack construction and deployment on power management integrated circuit and impacts on IoT applications. In: NDSS (2022)
38. Vrachkov, D.G., Todorov, D.G.: Research of the systems for firmware over the air (fota) and wireless diagnostic in the new vehicles. In: 2020 XXIX International Scientific Conference Electronics (ET), pp. 1–4. IEEE (2020)
39. Wara, M.S., Yu, Q.: New replay attacks on zigbee devices for internet-of-things (IoT) applications. In: 2020 IEEE International Conference on Embedded Software and Systems (ICESS), pp. 1–6. IEEE (2020)
40. Wee, Y., Kim, T.: A new code compression method for FOTA. IEEE Trans. Consum. Electron. **56**(4), 2350–2354 (2010)
41. Wu, Y., et al.: Your firmware has arrived: a study of firmware update vulnerabilities. In: USENIX Security Symposium (2023)
42. Zhang, Y., Li, Y., Li, Z.: Aye: a trusted forensic method for firmware tampering attacks. Symmetry **15**(1), 145 (2023)