AN EMPIRICAL STUDY ON THE SECURITY OF THE UPTANE STANDARD

Rareș-Mihail Visalom[1], Alin-Gabriel Antoci[2], Amelia-Ștefania Andronescu[3], Jan-Alexandru Văduva[4], Răzvan Rughiniș[5], Dinu Țurcanu[6]

Vehicles are increasingly becoming more complex and use a variety of smaller devices integrated deep within their architecture.

The devices that govern communication within and outside a car become outdated with the passing of time. In some situations, there are ways to update certain components, but only with physical interaction with the vehicle. In other situations, such practices are impossible.

As OTA Updates are becoming more and more present in the Automotive field, it is essential to be able to update the firmware of Electronic Control Units (ECUs) in a secure way. The current study formally explores the security principles of the Uptane Standard, highlighting the challenges that this field presents and focusing on the limitations and future work potential of the OTA Update field applied in the Automotive Industry. We also identify the need of integrating Secure Coding practices in the development process.

Keywords: OTA Updates, Uptane, Automotive, Cybersecurity, Threat Modeling

1. Introduction

Over-The-Air (OTA) Updates are a means of remotely updating a device. In the context of the Automotive Industry, this means being able to remotely update the firmware running on the Electronic Control Units (ECUs) inside of

[1]Doctoral student, National University of Science and Technology Politehnica Bucharest, Romania, e-mail: rares.visalom@upb.ro

[2]Masters student, National University of Science and Technology Politehnica Bucharest, Romania, e-mail: alin_gabriel.antoci@stud.acs.upb.ro

[3]Masters student, National University of Science and Technology Politehnica Bucharest, Romania, e-mail: amelia.andronescu@stud.acs.upb.ro

[4]Lecturer, National University of Science and Technology Politehnica Bucharest, Romania, e-mail: jan.vaduva@upb.ro

[5]Professor, National University of Science and Technology Politehnica Bucharest, Romania, e-mail: razvan.rughinis@upb.ro

[6]Associate Professor, Department of Software Engineering and Automatics, Technical University of Moldova, Chișinău, Moldova, e-mail: dinu.turcanu@adm.utm.md

a car. The update process usually involves, but might not be limited to: downloading the updates, checking their validity, applying the update, checking if the update was applied successfully, and recovering from a failed update.

Having a secure means of remotely updating cars would allow car manufacturers to patch vulnerabilities of their ECUs. For example, recent research discovered it was possible to unlock keyless entry cars [1] by exploiting a vulnerability. The code for such exploits is frequently publicly available, which adds great risk for the owners of such cars. With a secure update framework implementation, such problems would be patched in a relatively short period of time, without having to physically move the vehicle.

There are other use cases where being able to control the firmware of an ECU would help. Recent business practices of some car manufacturers involve enabling certain features of a vehicle based on a subscription model [2]. This obviously requires secure remote access to the car. Although the need for such an update framework is visible, a robust open-source implementation is still to be created.

The purpose of this paper is to provide a top-level view of the current state of the Automotive Over-The-Air Updates. The Uptane standard and its implementation are discussed in great detail in 2 and 3, respectively. Potential improvements of the Uptane standard are discussed in 4.

2. **Uptane**

The aim of this chapter is to digest and provide a summary of the most essential information of the Uptane standard. This implies that sometimes non-critical details will be omitted.

Uptane is not an implementation nor a working prototype, it is a standard that borrows concepts from The Update Framework (TUF) [3]. In other words, it is a set of directives and best practices of what a secure OTA framework geared towards the automotive industry should be. Implementations should respect these guidelines and implement a working system with security in mind.

Uptane adheres to a client-server architecture to deliver the updates. The client would be installed on a vehicle while the server would be hosted remotely.

2.**0**.1. *Purpose and preconditions of Uptane.* The purpose of Uptane is to provide a safe means of delivering updates to a client that would then update the ECUs inside a car. To this end, the Uptane standard implies the following preconditions, as detailed in Table 1.

Precondition number 1 implies that the car can access the server, which is expected in a client-server architecture.

Precondition number 2 might pose a security concern if the ECUs are directly exposed in the Internet. However, this is most likely not what the standard suggests. The "indirect" way of connecting, which implies a gateway between the Internet and the ECUs is a much safer approach. The overall

Table 1. Uptane Preconditions.

Nr	Precondition
1	Vehicles have network connectivity to the Server services
2	ECUs communicate with Servers directly or via a gateway inside the vehicle
3	ECUs are programmable and updatable
4	ECUs shall use public keys and perform hashing of images and metadata
5	Director and Image repositories shall be updated to the latest secure versions

architecture will be discussed in 2.**0**.2. Precondition number 3 implies flexibility on the ECUs. Flashing new firmware on hardware devices can sometimes lead to unexpected outcomes, which is why it is a challenging field [4]. Precondition number 4 implies a means for the system to check the integrity and validity of firmware images received from the server. This is essential to ensuring that updates have not been tampered while in transit over untrusted networks. Moreover, this also ensures the integrity of the system [5]. Precondition number 5 is the most general of the requirements and implies constant updates and maintenance of the most critical server-side components.

2.**0**.2. *Uptane Overall Architecture* . The current chapter will focus more on the interaction between the Uptane Servers and the vehicle. The Vehicle contains an ECU that is responsible with communicating with the Uptane Servers. This component is usually the Primary ECU, as having a separate ECU for the sole purpose of communication adds unnecessary complexity in most cases. Throughout the current paper, we will assume that the Primary ECU enables communication with the Servers.

Figure 1 provides a summary of the messages exchanged between the Primary ECU of a Vehicle and the Uptane Servers. The Uptane servers are the Director Repository, the Image Repository and the Time Server.

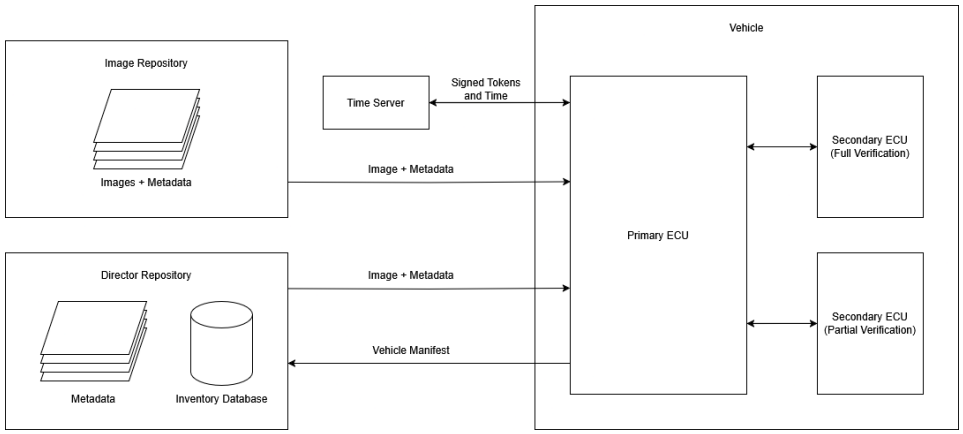

Fig. 1. Overview of the interaction between Uptane servers and a Vehicle.

The vehicle itself contains many ECUs, each with its specific purpose and mission-critical roles. In addition to very strict performance requirements, these ECUs must also operate in a very safe environment. Thus, the risks of altering or hindering the functionality of the ECUs are of the highest importance. Updates or changes must be verified for correctness and validated so that the vehicle can operate normally.

The architecture discussed will help in understanding the capabilities that an attacker has in the threat model discussed in 3. This chapter combines information discussed by the Uptane Standard in [6] and [7].

In the next sections the importance of the Software Repositories 2.**0**.3 and the ECUs 2.**0**.4 is discussed.

2.**0**.3. *Software Repositories .* This section focuses on the **Image Repository** and the **Director Repository**.

The **Image Repository** contains binary images to install on **ECUs** along with signed metadata about the images. Both the binary images and their associated metadata are provided to vehicles when vehicles request it. The images are provided by the OEM or suppliers and uploaded to the **Image Repository** whenever such operation is needed. Access to this repository requires authentication and authorization to prevent potentially malicious actions from propagating in the infrastructure.

The **Director Repository** provides signed metadata that instructs **ECUs** what images to install. This repository stores information about vehicles, **ECUs** and software revisions. Primary ECUs can upload vehicle manifests to the **Director Repository** and can also download metadata. Under certain circumstances, the **Director Repository** could encrypt images for **ECUs** that require them and it could either encrypt those images on-the-fly/on-demand or store encrypted images on the **Image Repository**.

To accomplish normal operation, **Director Repository** needs to include an **Inventory Database**. This needs to be a private database containing mappings between vehicles, ECUs and software revisions. In other words, the **Inventory Database** is a registry of relationships between all vehicles, the ECUs they contain and the firmware versions running on each of the ECUs.

The **Time Server** is another critical component that has the sole purpose of providing accurate system time. The timestamps are used during the metadata validation process.

2.**0**.4. *ECUs .* The **Primary ECU** is located inside the vehicle. It uploads vehicle manifests to the **Director Repository** to keep the **Inventory Database** updated. It also checks system time after having retrieved it from the **Time Server**, before it downloads and verifies the latest metadata. If the validation is completed successfully, it downloads and verifies the latest images for itself and for its **Secondary ECUs**.

The **Primary ECU** interacts with the **Image Repository**, **Director Repository** and **Time Server**.

The **Secondary ECU** verifies part of the information verified by the **Primary ECU**, namely time, metadata and images. However, as opposed to the **Primary ECU**, the **Secondary ECU** does not interact with the servers directly. It relies on the **Primary ECU** to provide all the information.

The **ECUs** can perform two types of verification processes, depending on their type:

- Full Verification: check metadata coming from both **Image Repository** and **Director Repository**.
- Partial Verification: only check the Targets metadata from the **Director Repository**, they do not check the metadata from the **Image Repository**.

The **Primary ECUs** can only perform Full Verification (never Partial Verification), while the **Secondary ECUs** can perform either Full or Partial Verification.

2.0.5. *Uptane Roles And Metadata .* There is another concept that goes hand in hand with the individual components discussed above, and that concept is Roles.

A Role is an entity that signs a metadata. The Role contains encryption keys to make signing possible. More details can be seen in [8].

Therefore, a Role can be thought of as just another component that ensures the Integrity of specific metadata.

One of the purposes of having multiple roles is to mitigate risk. Having a `Single-Point-Of-Failure` would mean that if the component is compromised, the entire infrastructure is compromised as a consequence.

Individually, the Roles have more or less the following purpose:

Root Role: This is the Certificate Authority of the Uptane system. It produces and signs Root Metadata.

- **Root Metadata:** Mapping between all four roles (Root, Targets, Snapshots and Timestamp) and their associated public keys.
- **Purpose of Root Metadata:** Manages (distributes and revokes) public keys used to verify the Root, Timestamp, Snapshot, and Targets role metadata.

Targets Role: Produces and signs Targets Metadata.

- **Targets Metadata:** All relevant information (filename, hash and file size, might contain compatible hardware for the current image) of images **to be installed** on ECUs, for each Image necessary for a client ECU. Might also contain delegation information, but this topic is not discussed in this paper.
- **Purpose of Targets Metadata:** Used to verify Images.

Snapshot Role: Produces and signs Snapshot Metadata.

Snapshot Metadata: All relevant information (filename and version number) of the Targets Metadata file, for each Targets Metadata file necessary for a client ECU.

Purpose of Snapshot Metadata: Used to specify which bundle of images were released at a point in time.

Timestamp Role: Produces and signs Timestamp Metadata.

Timestamp Metadata: All relevant information (filename, hash(es) and version number) of the **latest** Snapshot Metadata file.

Purpose of Timestamp Metadata: Indicates whether there is any new Metadata and/or Images.

A very important mention is that the roles above exist on both the Director Repository and the Image Repository. Moreover, Roles do not share the same keys between Repositories. As an example, the Root Role from the Director Repository will not have the same key as the Root Role from the Image Repository.

The same Roles [9] are also employed by The Update Framework (TUF) [3], from which Uptane borrows various concepts.

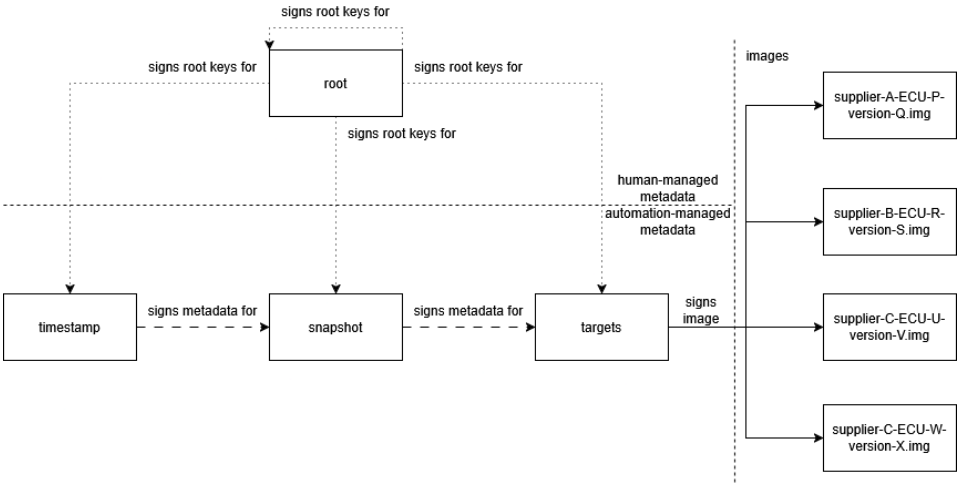

Fig. 2. Configuration of Roles on the Director Repository.

Figure 2 provides an overall look over what the Roles perform. The Root Role will sign the Metadata of the Root (itself), Timestamp, Snapshot and Targets role. This is the basis of safe communication, as this process establishes the keys used to check the signatures emitted by all of the Roles.

The management of the Root Role is human-controlled in the sense that keys will be initialized and updated (revoked and new ones issued) after human intervention.

The interaction between the Timestamp, Snapshot and Targets Roles is different in comparison to the interaction of the Root Role with all the other

roles. In other words, while the Root Role distributes keys, the other three roles use the keys.

The internal Metadata structure will be abstracted throughout this paper by the following simple model: payload (usually filenames, file sizes and file hashes) and signature (validates the above payload).

The purpose of signing the Metadata is to ensure **integrity** of the information via a chain of trust: Timestamp Role signs Snapshot Metadata, Snapshot Role signs Targets Metadata and Targets Role signs Image Metadata. This "chain of trust" is validated at each step using the Keys managed by the Root Role.

Notice that the Images themselves are not contained within any of the Metadata structures. The images are only downloaded once all the Metadata Structures are validated (see 2.**0**.6). This validation process is the main strength of Uptane (and of The Update Framework (TUF) [3]).

2.**0**.6. *Uptane Update Process And Full Verification .* The activity of a Primary ECU will be detailed to highlight the update process.

In summary, the actions are: Download, Verify and distribute Time, Metadata and Images. The individual steps performed by the Primary ECU are outlined in Table 2.

Table 2. Uptane Primary ECU Actions.

Nr	Action
1	Send vehicle manifest to Uptane Servers
2	Download and validate current time
3	Download and validate metadata
4	Download and validate images
5	Send time information to Secondary ECUs
6	Send metadata to Secondary ECUs
7	Send images to Secondary ECUs

As described in 2.**0**.2, the Primary ECU is connected to all Secondary ECUs. The Primary ECU will be the only component that interacts with the Uptane Servers, since Secondary ECUs do not communicate directly over the Internet. The current discussion will focus on steps 2, 3 and 4 of the list above.

The Primary ECU will perform Full Verification of Metadata. This means that the Primary ECU will check Metadata coming from both the Director Repository and the Image Repository. Apart from being valid, the information from the two sources should match and not contradict itself in any way. The Full Verification steps are summarized and explained below. The explanation divides the original verification process detailed by the standard [10] and annotates it to improve clarity where necessary:

(1) Get current time.
(2) Download and Validate Root Metadata from Director Repository.

(3) Download and Validate Timestamp Metadata from the Director Repository.
(4) Extract file hash and version number of the Snapshot Metadata from within the Timestamp Metadata. Compare the file hash and version number with those of the cached version of the previous Snapshot Metadata (if available).
    - If the new Snapshot Metadata is the same as the previous one, the verification process is stopped and marked as Complete.
    - If the new Snapshot Metadata is different and more recent, proceed to next step.
(5) Download and Validate the most recent Snapshot Metadata from the Director Repository.
(6) Download and Validate Targets Metadata from Director Repository.
(7) Extract file name, file hash, file size (and possibly other) information from the Targets Metadata.
    - If the extracted information indicates that there are no updates to perform, the verification process is stopped and marked as Complete.
    - If there are updates to perform, proceed to next step.
(8) Download and Validate Root Metadata from Image Repository.
(9) Download and Validate Timestamp Metadata from the Image Repository.
(10) Extract file hash and version number of the Snapshot Metadata from within the Timestamp Metadata. Compare the file hash and version number with those of the cached version of the previous Snapshot Metadata (if available).
    - If the new Snapshot Metadata is the same as the previous one, skip to the last step.
    - If the new Snapshot Metadata is different and more recent, proceed to next step.
(11) Download and Validate the most recent Snapshot Metadata from the Image Repository.
(12) Download and Validate Targets Metadata from Image Repository (the top-level Targets Metadata).
(13) Compare the Targets Metadata from the Director Repository with the Targets Metadata from the Image Repository. They should match. The Primary ECU should perform this check for each of the images listed by the Targets Metadata downloaded from the Director Repository in step 7. Secondary ECUs can perform this check only for the image it will install (the image is identified by the Image Metadata from the Director Repository that matches the ECU identifier of the current ECU). The detailed steps of comparing the Targets Metadata are:

(a) Locate and Download a Targets Metadata from the Image Repository that contains an image with the exact filename as the one listed in the Director Metadata.
(b) Comparison between Targets Metadata from Director Repository and Targets Metadata from the Image repository:
    (i) Compare noncustom metadata (file length, file hashes) of either the encrypted or unencrypted images. They should be the same in both sets of Targets Metadata.
    (ii) Compare custom metadata (hardware identifier, release counter if present). They should be the same in both sets of Targets Metadata.
    (iii) Validate the release counter, if present. The release counter found in the previous Targets Metadata file is less than or equal to the release counter found in the current Targets Metadata. Applies for both the (old Director Repository Targets Metadata, new Director Repository Targets Metadata) and (old Image Repository Targets Metadata, new Image Repository Targets Metadata) pairs.

There is a slight ambiguity in 13(b)iii about checking the release counter. It was not clear whether the counter in the old Director Repository Targets Metadata and the counter in the new Director Repository Targets Metadata would be compared along with the same check being performed on the Targets Metadata coming from the Image Repository. The ambiguity is solved in [11].

3. **Limitations And Threat Model**

The main purpose of Uptane is the secure delivery of updates. Hence, preventing supply chain attacks was not a focus of Uptane at the beginning. However, subsequent work has been done and support for ensuring a secure supply chain can be built into an implementation of Uptane [12] [13].

Two relevant Out-Of-Scope elements listed by the standard in [14] are Injection of malware in trusted packages and rogue package repository mirrors, which would contain malicious packages matching the original packages.

The limitations are consistent with the idea that Uptane deals only with secure **delivery** of an update and does not secure any other part of the full process of developing, distributing and applying an update.

However, to ensure the overall security of the system, guidelines detailing the interaction between Uptane and the other areas of the update process (e.g. secure development and secure application of updates) are required.

Essentially, both out-of-scope elements state that Uptane does not perform malware checking of images in any way. Nevertheless, this is expected from a framework that deals only with the delivery of such updates. We stress the fact that a system that implements the Uptane standard should implement

such practices. The Attacker Goals [15] are a key component of the Threat Model elements. The goals are summarized in Table 3.

Table 3. Attacker goals summarized.

Nr	Attacker Goal
1	Intercept update contents and attempt to reverse-engineer it
2	Prevent installation of updates
3	Stop one or more ECUs inside the vehicle
4	Control ECUs within the vehicle

The attacker should not be able to compromise a vehicle. Compromising a vehicle involves compromising the Uptane Server or the communication between the Uptane Server and the Vehicle. In extreme cases both could be compromised, as this would guarantee that potentially malicious updates reach the Vehicle or that valid updates don't even reach the Vehicle.

Being able to either crash or control an ECU is what would be labeled as a success from the Attacker's point of view. The Attacker is not necessarily concerned with persistent control of the compromised ECUs, nor with constant remote access into a vehicle (although this would also be a critical security flaw and a tremendous concern).

Even blocking newer updates from reaching the target ECU is a valid objective for the attacker, as this might ensure that older, unpatched or vulnerable, versions of firmware might continue to run on ECUs.

The goals reflect scenarios that an attacker might want to successfully perform. In addition to the goals, the Threat Model contains Attacker Capabilities [16], or actual instruments that the attacker is assumed to be able to use to try and compromise the overall security of the system. In summary, the attacker capabilities are Man-In-The-Middle attacks (intercept and alter network traffic), Compromise either the Director Repository or the Image Repository (only one of the two) and Compromise either the Primary ECU or a Secondary ECU (only one of the two).

Even in such concerning circumstances, the Update Standard should be resilient enough to at least identify the threat, so that recovery would then be possible. It is important to note that the Uptane Standard is prone to the same risks [17] as The Update Framework (TUF) [3].

Uptane is a standard and not an implementation, consequently an implementation of it is required for a practical assessment to be performed against it. The first implementation of the Uptane Standard [18] was a reference implementation [18] that is now obsolete. The obsolete reference implementation brought great value as it was not only the active embodiment of the Standard, but also served in assessing the overall security of the ideas. The obsolete reference implementation was the basis of the first Penetration Test performed against an Uptane implementation [19].

At the time of writing this article, the best and most convenient implementation available is the latest version of `ota-community-edition` [20]. The project is open source and it supports research in the Automotive Software Over-The-Air Updates field. To aid in research efforts, we provide a step-by-step guide to setting up a working infrastructure in [21]. We abstracted the hardware requirements of setting up such an infrastructure. The official documentation is limited in this regard and does not offer recommendations on hardware requirements, which might pose a technical difficulty to researchers. This topic could be the subject of future research and could quantify how hardware needs increase as the number of client Vehicles increases.

Our research identified another concern, which might also constitute a complex future work project: the lack of open source alternatives for OTA Update distribution frameworks for Automotive.

The current research identifies the need to implement better security practices in the process both developing and then testing OTA Update frameworks. As a general recommendation to ensure the security of any implementation of the standard, Security Testing must be implemented as soon as possible in the Software Development Life Cycle (`SDLC`) to maximize the chances that flaws being detected before progressing further in the pipeline. The means of implementing Security Testing could be Static Application Security Testing (`SAST`) over the codebase, Dynamic Application Security Testing (`DAST`) over the latest build generated from the latest codebase and manual Penetration Tests performed by experts against the latest deployment of the latest build.

A more practical future work project that the authors are working on is a practical Penetration Test performed on the latest ota-community-edition version. The work will be detailed in a future paper and will be based on an assumed breach scenario of an attack originating inside the network perimeter.

4. **Conclusion**

Our work provided a formal evaluation of Uptane's cybersecurity posture. We discussed the inner structures of Uptane Standard, with a focus on its core Metadata verification process and the chain of trust that prevents having a Single-Point-Of-Failure. We also highlighted some limitations and the Threat Model that Uptane was designed to protect against. Although the Uptane standard might be robust enough from a formal perspective, it still needs to be implemented and deployed. Security tests of the implementation should also be performed before releasing new features. To this end, having a common, general, set of best practices to secure the development process of future Uptane Standard implementations is highly required. This could enable developers to discover vulnerabilities early during the development process.

Acknowledgement

This scientific research is financially supported within the project 'System for Scanning and Mapping IP Resources in Romania for the Early Detection of Cyber Threats,' contract no. 25Sol(T25)/2024, funded under the PN IV Program, 5.6 – Challenges, Subprogram 5.6.3 – Solutions.

REFERENCES

[1] RollingPwn. Rolling pwn attack. [Online]. Available: https://rollingpwn.github.io/rolling-pwn/
[2] T. Verge. Bmw starts selling heated seat subscriptions for $18 a month. [Online]. Available: https://www.theverge.com/2022/7/12/23204950/bmw-subscriptions-microtransactions-heated-seats-feature
[3] T. U. Framework. The update framework. [Online]. Available: https://theupdateframework.io/
[4] S. El Jaouhari and E. Bouvet, "Secure firmware over-the-air updates for iot: Survey, challenges, and discussions," *Internet of Things*, vol. 18, p. 100508, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660522000142
[5] S. Brightwood, "The importance of secure firmware updates in maintaining system integrity," 09 2024.
[6] Uptane. Detailed design of uptane. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#5-detailed-design-of-uptane
[7] ——. Server / repository implementation requirements. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#53-server--repository-implementation-requirements
[8] ——. Uptane role terminology. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#22-uptane-role-terminology
[9] T. U. Framework. The update framework roles and metadata. [Online]. Available: https://theupdateframework.io/docs/metadata/
[10] Uptane. Uptane full verification. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#5442-full-verification
[11] ——. Uptane custom metadata about images. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#52311-custom-metadata-about-images
[12] U. S. Group. Scudo: A proposal for resolving software supply chain insecurities in vehicles. [Online]. Available: https://uptane.org/papers/scudo-whitepaper.pdf
[13] Uptane. Integrating software supply chain security into uptane. [Online]. Available: https://uptane.org/docs/latest/deployment/best-practices#802-integrating-software-supply-chain-security-into-uptane
[14] ——. Out of scope. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#34-out-of-scope
[15] ——. Attacker goals. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#41-attacker-goals
[16] ——. Attacker capabilities. [Online]. Available: https://uptane.org/docs/2.1.0/standard/uptane-standard#42-attacker-capabilities
[17] T. U. Framework. Security properties of tuf repositories. [Online]. Available: https://theupdateframework.io/docs/security/
[18] Uptane. obsolete-reference-implementation. [Online]. Available: https://github.com/uptane/obsolete-reference-implementation