Trusted Update System Overview

System Software Development Department,

InfoTech,

Connected Advanced Development Division,

Toyota Motor Corporation

Contact : Okino, Naoto (naoto.okino@toyota-tokyo.tech)

Revision History

Version	Date	Description
0.1	24th, December, 2021	Initial Releaase
0.1.1	14th, January, 2022	Correction of errors in spelling and grammar, Font integration and Update Copyright

Introduction

This document describes the overview of the Trusted Update System.

Background

In general, software and data update systems are developed and optimized for each individual device. Therefore, there are few update platforms that are used broadly among various devices.

In terms of general update platform systems, there are many difficulties such as various and complex hardware specifications of target devices. In addition, each device also might have a specific feature about its update.

For example, in a case of automobile update,

- There are multiple distributed systems inside it, and the combination of each update must be managed strictly.
- The system structure also varies greatly depending on models.
- Automobile software and data updates must comply with governments' regulations since it might affect the safety of the product.

It is not easy to realize a generalized system by abstracting differences among many devices while resolving each device’s specific issue such as one described above. However, considering the recent increase of connected devices, the demand for the generalized update platform is supposed to be increasing.

Our Vision

Each developer and organization can use a generalized update platform and is able to realize stable and high-performance updates with low cost and high efficiency.

Our Mission

We develop a generalized update system of software and data, which can be used for various devices and will be a de-facto standard of update system in the future.

System Overall

TUS Abstraction

The generalized update system we'll provide is called Trusted Update System (TUS). Fig.1 shows the abstract of the TUS. The TUS is able to control updates of various devices regardless of the delivery methods.

In addition, the JIT-DF system(Just-In-Time DataFlow), which is able to control many devices with commands. The JIT-DF system has functions such as EdgeAI, but the TUS uses what the JIT-DF is customized for update processing as the TUS runtime.

A flexible update system that can fit cloud-side operation and target device specification can be established by using the JIT-DF system. For example, Just-In-Time and Push styled update systems can be realized by publishing a script which downloads an update package hosted at a specific CDN. In addition, the JIT-DF system can be used also as Telemetry data collection.

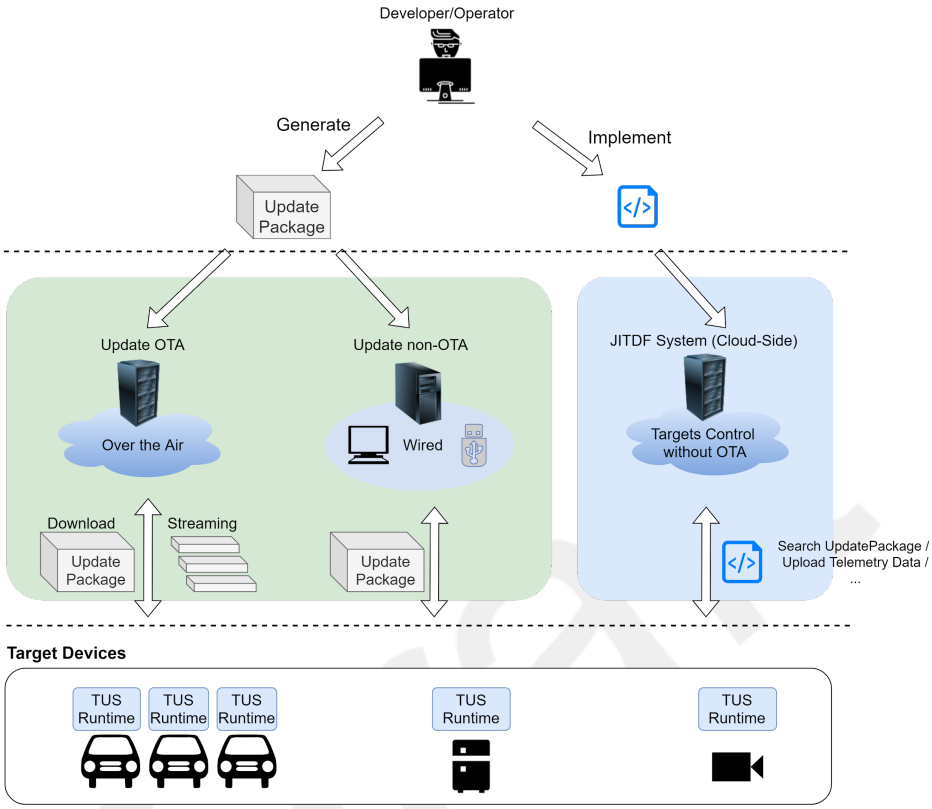

Fig. 1 Abstract of update using TUS

The TUS also provides Host-Tool to generate packages flexibly depending on devices for developers in a way shown in Fig. 2.

As Fig. 2 shows the abstract , the Package Generator of this Host-Tool has a function to compress, encrypt and generate scripts to control complex updates in a device (details in later).

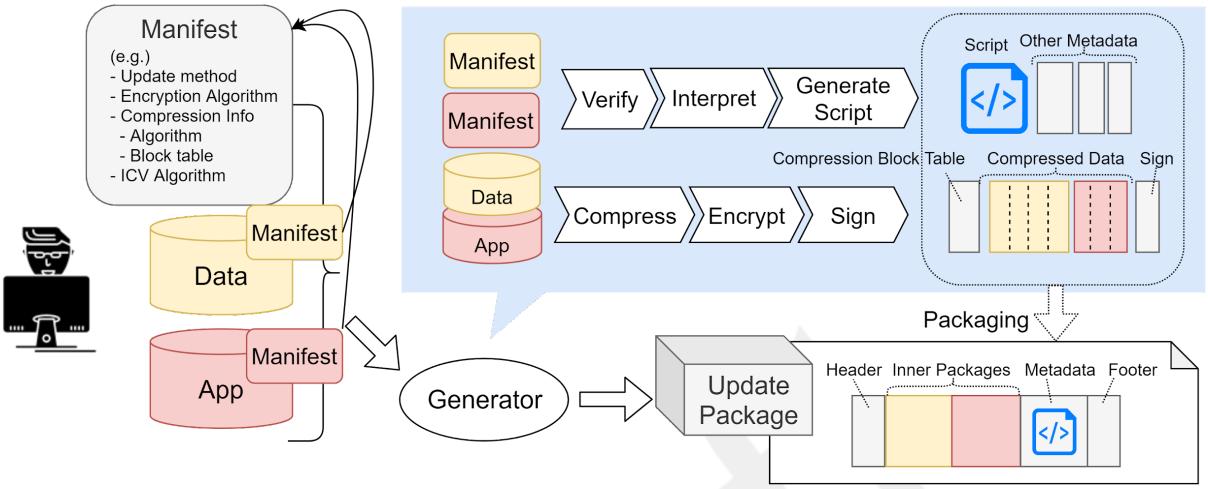

Fig. 2 Abstract of Package Generation

Fig. 3 shows the abstract of the inside of Target Device. In the TUS, Target Device is not always a single physical device. A physical, simple structure device can be Target Device, and products such as an automobile containing multiple ECUs also can be Target Device.

The TUS Runtime contains the Package Parser and scripts runtime. Any updates of complex system/software structured devices can be realized by these scripts.

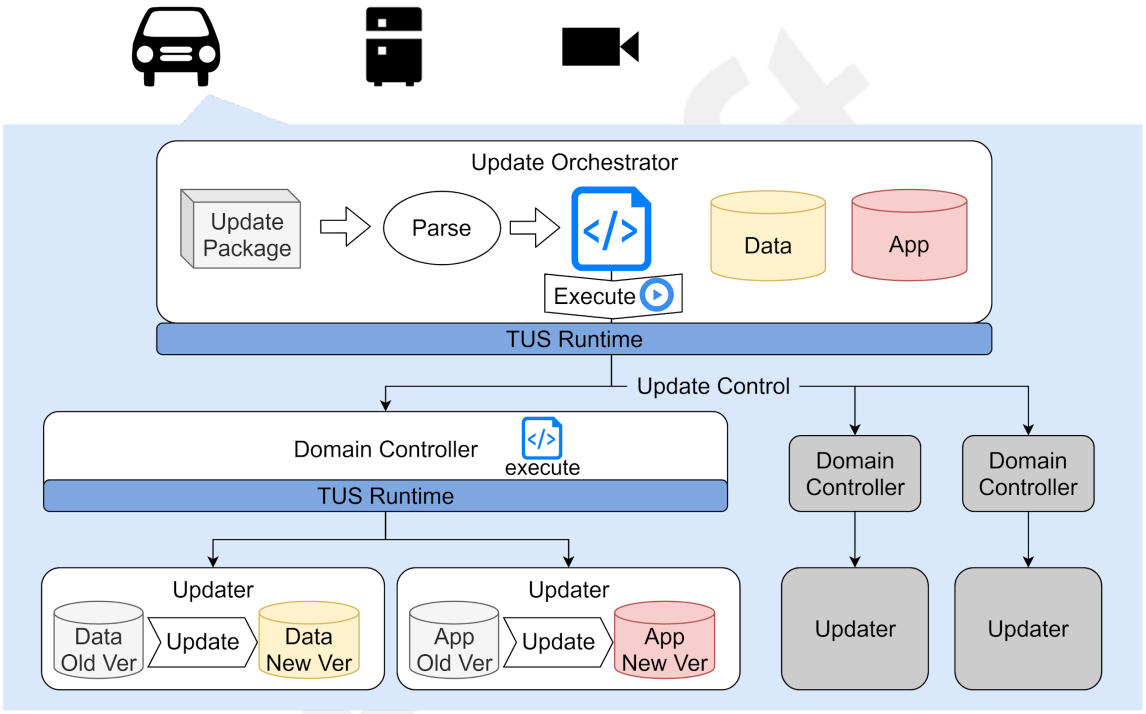

Fig. 3 Abstract of Update Control in devices

Various software and data can be updated

The update target of the TUS is described before deep diving into the details.

Update Target

Update Target is the target itself that is updated. Any digital data on computer systems can be Update Target. Therefore, software(e.g. OS, Apps) and data(e.g. map data, images and videos) can be defined as Update Target.

Updater

Updater is the software component that updates Update Target. There is not always only one Updater for a single Update Target type. Multiple Updater for a single Update Target type can be defined such as normal Updater and Delta-Updater (Delta Update) depending on the necessity.

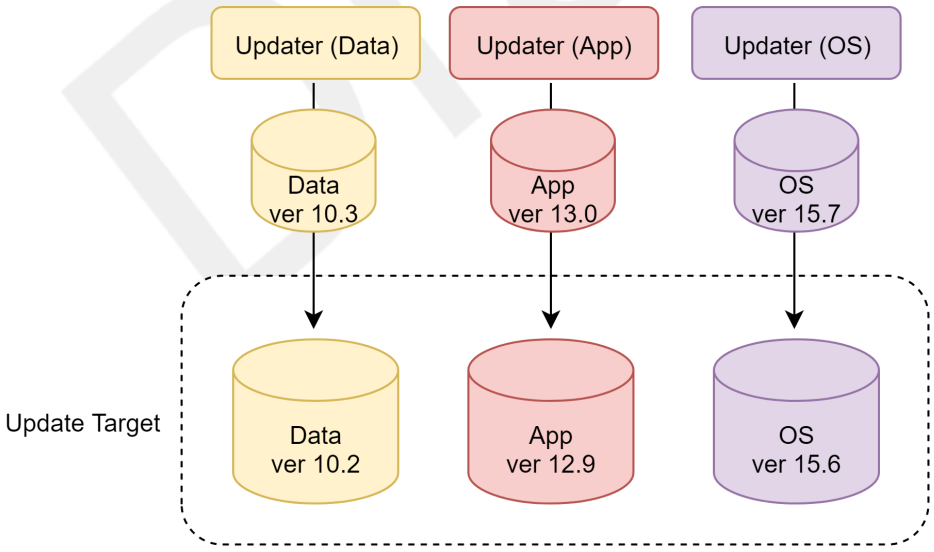

Fig. 4 Update Target and Updater

Distributed management model for updates of any devices

The Target Device might have various internal system structures and Update Target itself also varies greatly. Especially, devices such as automobiles consist of multiple ECUs of various hardware functions and software stacks, and the whole system must be updated all at once without any inconsistencies. In addition, various Update Target types such as multiple children ECUs of a specific ECU, environment with micro-computers and multiple OS hosted by hypervisor can also exist.

Fig. 5 shows dependency relationships of an automobile example. There are dependencies both inside an ECU and between ECUs, and a specific Update Target must be rebooted after update. It is necessary to control update order and roll-back when an update fails, and system down-time must be as short as possible in terms of UX.

It is not realistic to control these complex system updates with a simple structure like a single update manager and multiple updaters.

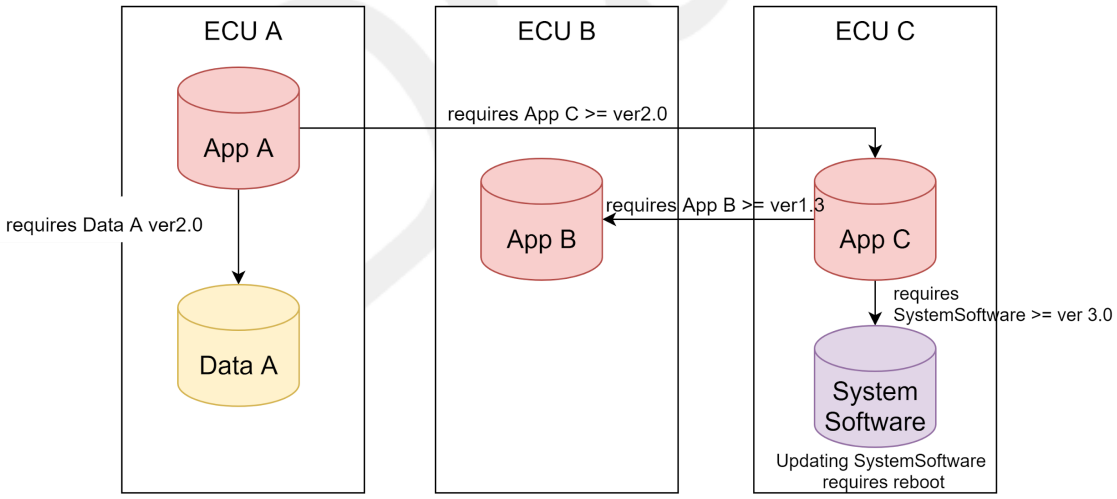

Fig. 5 Complicated update dependency

In order to generalize and abstract these various and complex updates, we define a 3 layered distributed management model of update related functions.

1. Updater
   The responsibility is to update each Update Target.
2. Domain Controller
   The responsibility is to control Updaters belonging to logical groups called Domains.
3. Update Orchestrator
   The responsibility is to arbitrate updates of the whole system, manage progress and communication with external systems (e.g. package downloading, notifying update result and coordination with UI).

Fig. 6 shows the abstract of the distributed management model. For example, a distributed management model model of an automobile might have an Update Orchestrator in a Central Control ECU, and ECUs with OS and computing resources such as IVI could be Domain Controllers, and microcomputers connected with IVI could be Updaters. Each 1-3 components must exist in any small Update Targets.

Following each section describes these components details.

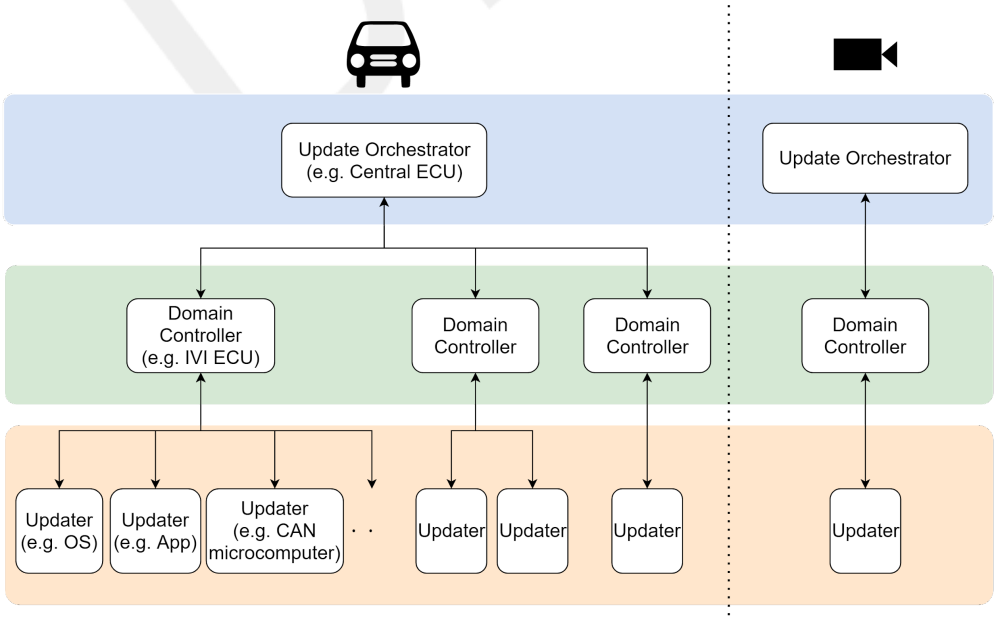

Fig. 6 Distributed Update Management Model

Domain

As described above, it is difficult to control and update complex systems with a lot of Updaters with a single component like Update Manager. Therefore, a logical group that can contain multiple Updaters, called Domain, is defined. As well as Updaters, Domain can be defined flexibly depending on platforms. Simple structured devices might have only one Domain, but complex structured systems which consist of multiple components connected with each other can be updated by multiple and optimized Domains.

Domain Controller

Components which control Updaters inside Domains are called Domain Controllers. Update Package contains scripts for each Domain, which describe update methods of Update Targets. Domain Controller will be a script runtime and execute them.

Domain and Domain Controller can be defined flexibly depending on hardware/software structures as described below. For example, Fig. 7 shows Domain A containing multiple physical computers(not only physical devices but also multiple platforms on hypervisor can exist) and a single Computer B containing multiple Domains inside it.

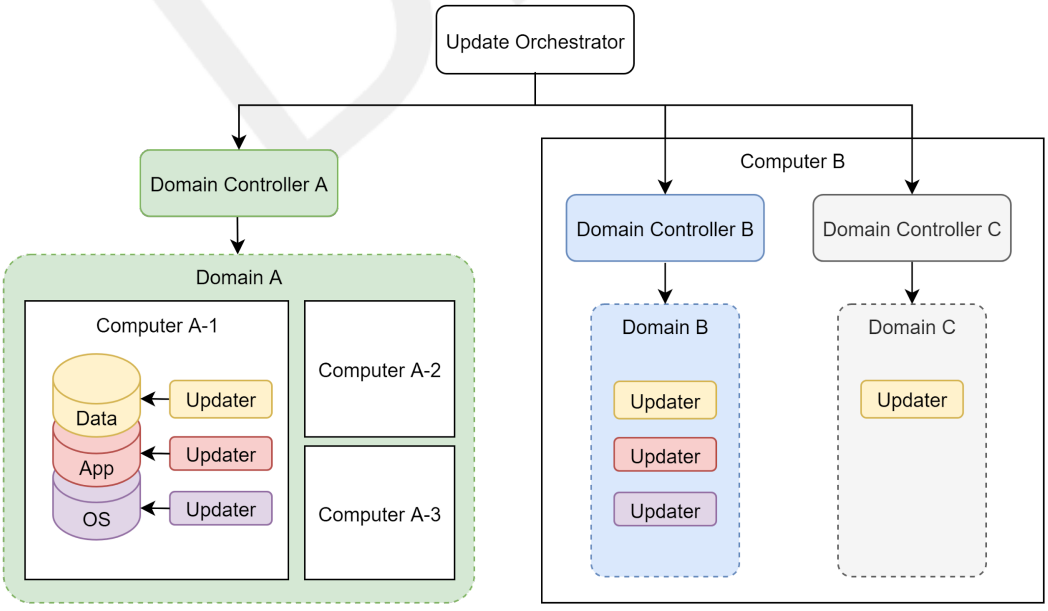

Fig. 7 Domain, Domain Controller and Update Orchestrator

Update Orchestrator

Only one Update Orchestrator exists in each Target Device. It manages updates of the whole device, arbitrates Domains to meet each constraint, communicates with external systems, finds/obtains update packages and notifies update status. As well as Domain Controllers, Update Orchestrator will be a script runtime and execute the scripts to control updates.

In particular, Update Targets might have complex dependencies that make it difficult to control updates properly. In the example of Fig. 8, App A depends on both Data A and App C in another Domain, and App C also depends on App B across Domains. Update Orchestrator manages the progress of updates involving these dependencies among Domains, and controls updates appropriately and efficiently.

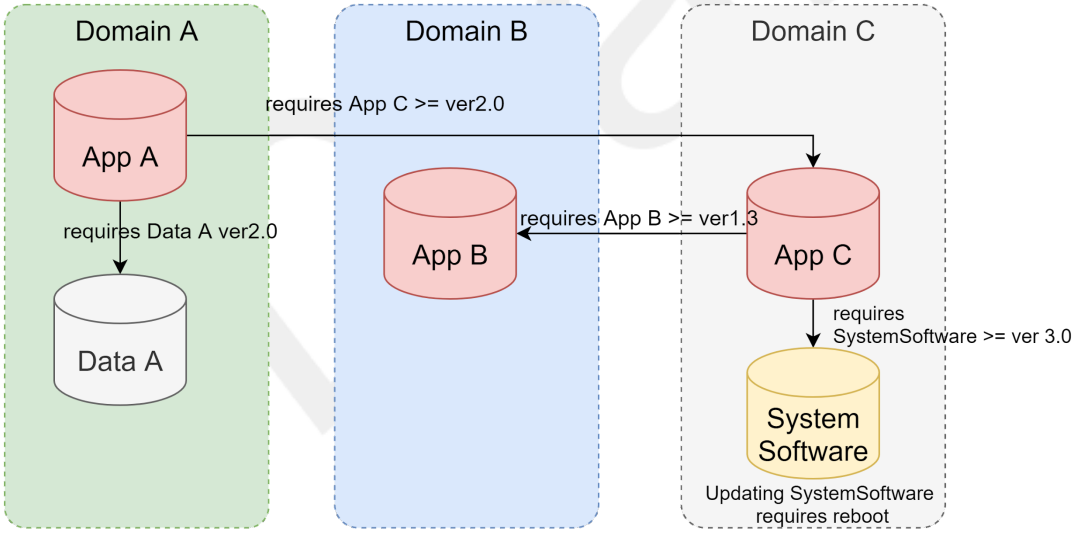

Fig. 8 Version Dependency

Update control using scripts

There are the following difficulties in updates using static programs that are natively implemented and embedded.

- Cannot control of complex update sequences
  In order to continue updating a wide variety of devices, it is necessary to control the update order among multiple domains and the rollback in case of update errors while taking into account the version constraints among multiple Update Targets. As devices evolve with updates, the dependency between Update Targets may become more complex, and the update sequence may become more complicated. Therefore, it is not realistic to continue to support this with only static programs.

- Increasing cost for large number of Update Target combinations
  When there are countless Update Targets in a Target Device, the number of combinations increases explosively in proportion to the number of Update Targets. Among these combinations, there are some combinations of Update Targets that are operationally impossible, but if the program is embedded statically, all combinations and control logic must be implemented and tested.

Therefore, as shown in Fig. 9, the TUS provides an update control mechanism using script language instead of a static update method. A script describing the update control is implemented when the Update Package is generated, and distributed in the Update Package. On the target device side, Update Orchestrator and Domain Controller will be runtime for the scripts, and control the Updaters in its Domain.

In this way, by not embedding the update control logic inside the target device, it is possible to evaluate only actual combinations which will be used. In addition, even if the update sequence becomes complex, it is possible to implement a script to control it.

This script needs to be created at the same time as the Update Package, and the TUS SDK provides a mechanism to generate this script. For the specific configuration on the target device, please refer to TD-1.

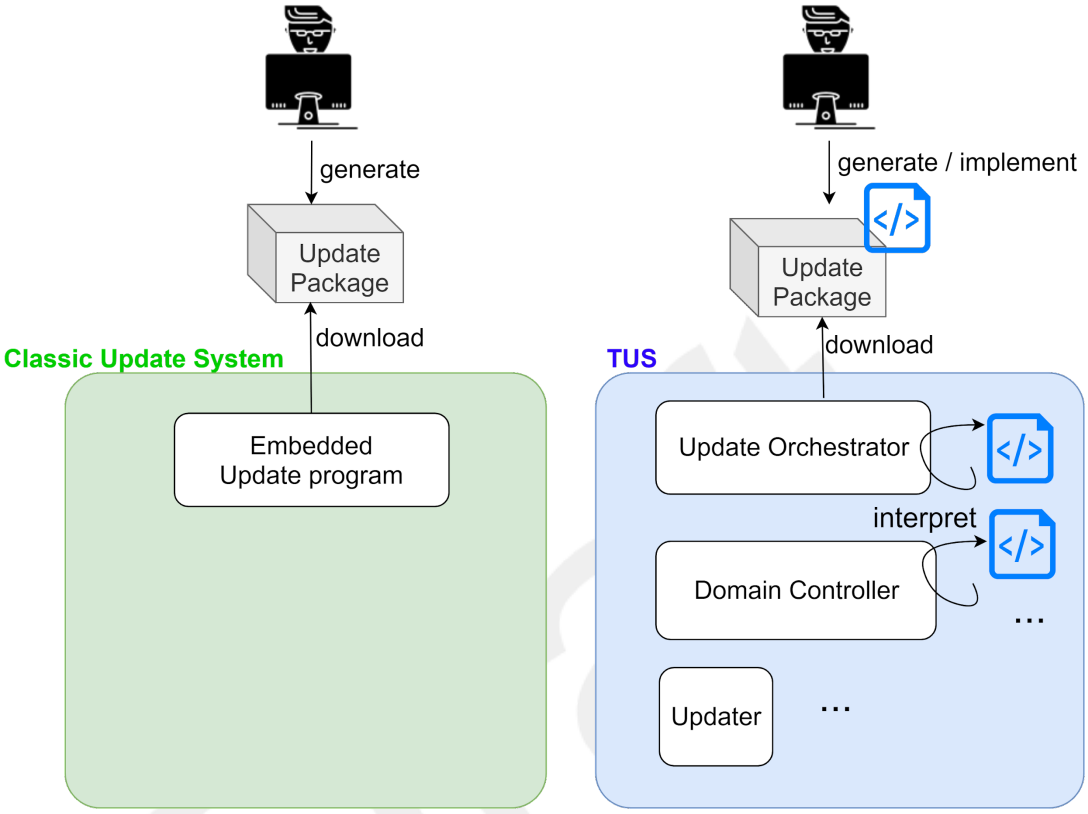

Fig. 9 Comparison of Update System

Generalized version management model

Updates are always performed based on the version of the Update Target. In addition, there are cases where it is necessary to assign a specific version to an Update Target Group consisting of multiple Update Targets, rather than to each Update Target. In addition, there are cases where a specific version is required for the entire system, such as version corresponding to some regulations, and therefore it is necessary to assign a version to multiple Update Target Groups and Update Targets.

Therefore, the TUS provides a general-purpose version management model that can assign versions to the entire system including not only multiple Update Targets, but also a logical Update Target Group that binds them together. This makes it possible to operate Domain and Update Target Group in the same manner as described above, operate a Domain that contains multiple Update Target Groups, and to assign specific versions to the entire Target Device.

For example, Fig. 10 shows an example of this version management.

- Each OS A and App A has their version.
- Version of Update Target Group A includes OS A and App A. This might be necessary to synchronize updates of OS A and App A because of government regulations.
- Update Target Group B also defined as well as Update Target Group A.
- Update Target Group X contains Update Target Group A and Update Target Group B. This management model can have a hierarchical structure of versions.
- App X, App Y and App Z are not included in any Update Target Group. This means App X, App Y and App Z can be updated independently on the other Update Targets.

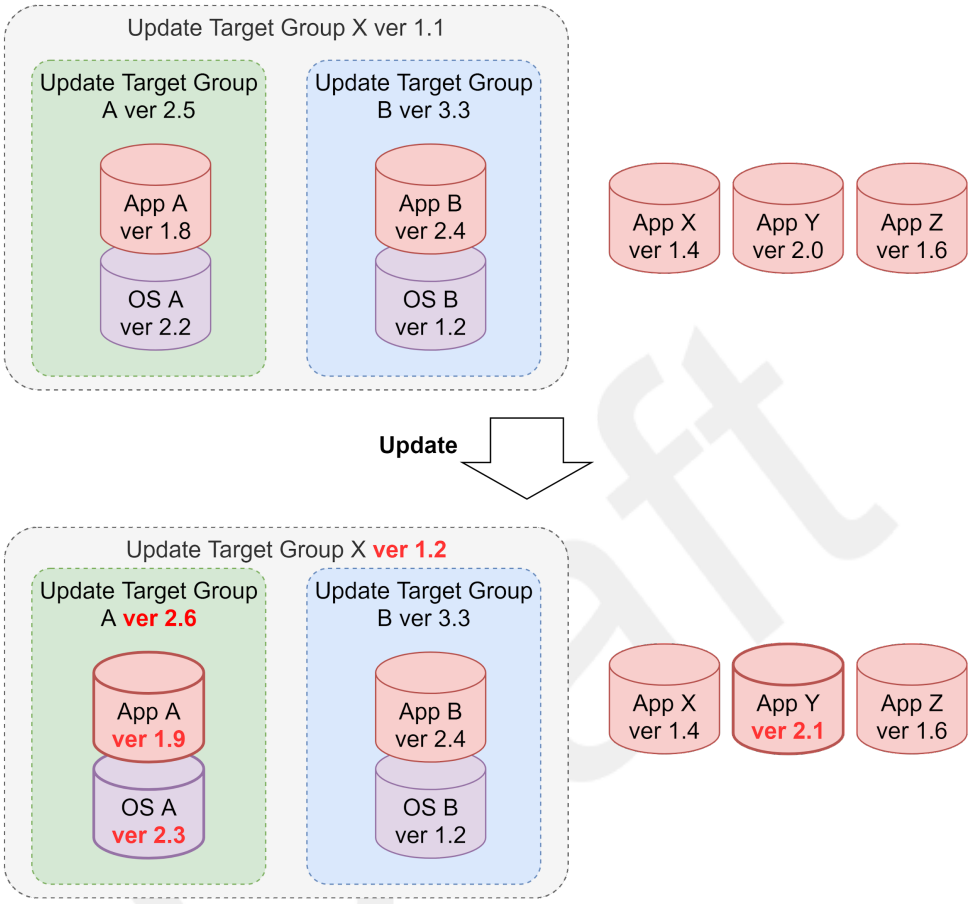

Fig. 10 An example of Update Target version management

Version evaluation function

Some Update Targets might have system-specific versioning models such as string comparison instead of incremental versioning. Also, in order to perform an update, it must be able to evaluate whether it is a version up or a version down. Therefore, the TUS provides a framework that can apply its own version evaluation function. By using this framework, it is possible to evaluate version ups and version downs in the TUS while keeping the system's original versioning model.

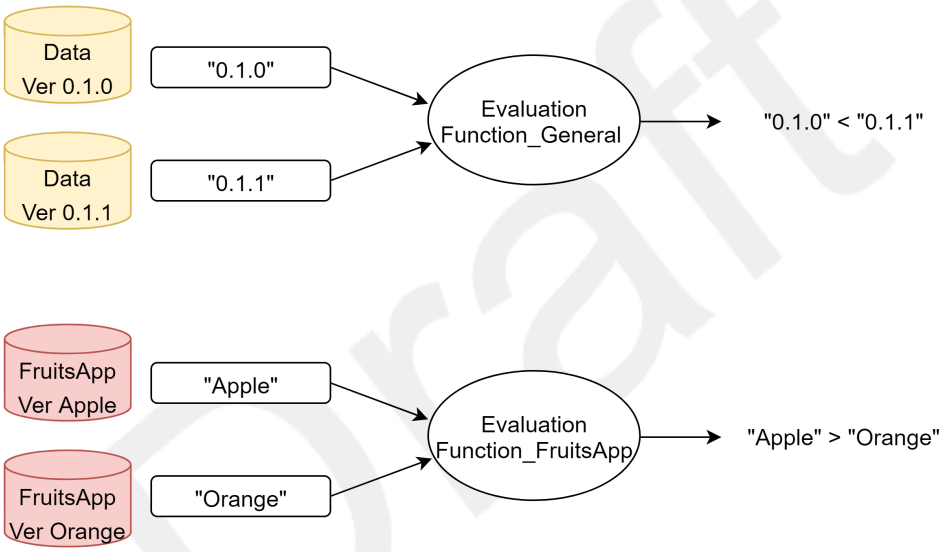

Fig. 11 Examples of Evaluation Function

Minimum Required Version and Beta Version

Depending on Update Targets, there may be cases where there are constraints that require a certain version or later to be used as a service (minimum required version), or cases where a beta version is provided that can be used by all or specific users upon confirmation of their intentions. In order to deal with these cases, the version control information can be included in the Manifest described below, and updates can be controlled accordingly. The Update System is also linked with external functions such as the UI to confirm the user's intentions. (Different version control can be applied for developers than for general users. The version order shown in the figure is just an example, and can be freely defined by the Update Target).

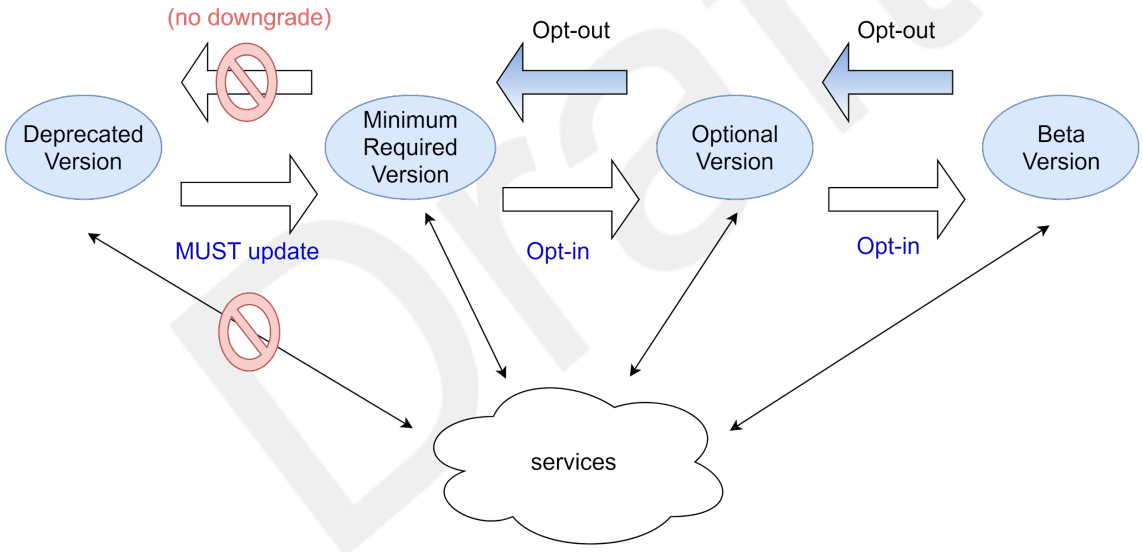

Fig. 12 Example of Minimum Required Version and Beta Version management

Delta Update

(TBD due to patent investigation)

Telemetry system to obtain data without software updates

It is assumed that demands for various metrics data of Target Devices will be changed or added after it is put into operation in the market. This system provides a telemetry system to acquire the necessary data on the target device, which can be immediately updated without software updates, thus realizing agile and flexible data acquisition.

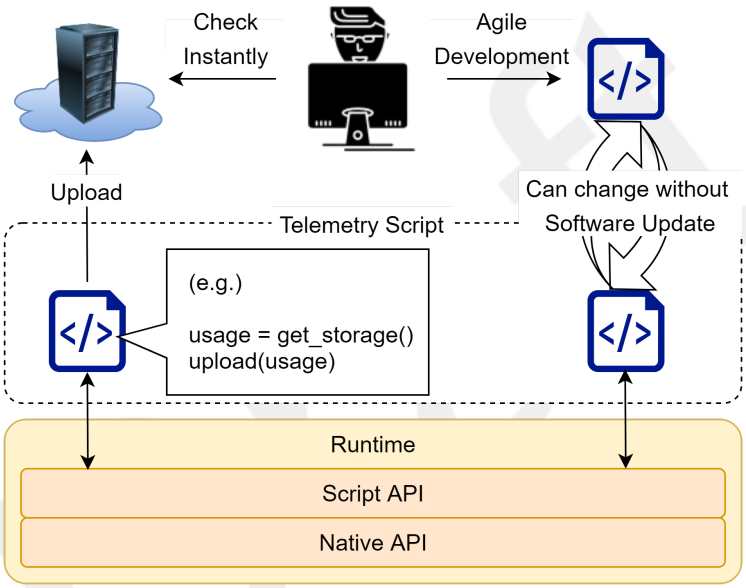

Fig. 13 Updatable Telemetry Scripts

The SDK for this telemetry system will also be included in this system, allowing each developer to develop and test telemetry scripts.

A cloud-side system to deliver and update the scripts to each device will also be included in the development.

Feature List

This list describes the features of the TUS.

Table. 1 Feature List

Group	ID (Link)	Feature
Overall	O-1	Various software and data can be updated
	O-2	Distributed management model for updates of any devices
	O-3	Update control using scripts
	O-4	Generalized version management model
	O-5	Delta Update
	O-6	Telemetry system to obtain data without software updates
Update Package	UP-1	Flexibility allowing to add data/metadata in any format (multiple and proprietary format packages can be contained)
	UP-2	Free allocation of data/metadata
	UP-3	Apply to devices with fewer resources (configuration for each device)
	UP-4	Partial process without waiting for the whole data
	UP-5	Structure to extract/deliver only necessary data
	UP-6	Metadata format to realize forward/backward compatibility
	UP-7	Tamper/damage check at every package read
	UP-8	Tamper/damage check of only necessary part without a whole data
	UP-9	Encryption, sign and compression for each data range depending on requirements (Security requirements such as encryption strength and algorithms and device constraints)
	UP-10	Protecting and obfuscation of update content
Target Device	TD-1	Flexible update control using script
	TD-2	Abstraction layer to reduce porting cost
	TD-3	Using hardware accelerator
	TD-4	High speed process on Target Devices with sufficient resources

	TD-5	Using hardware security functions
	TD-6	Integration with external systems (UI/Target status management server)
SDK	SDK-1	A set of tools to apply to various development flows and environments
	SDK-2	High speed generation of large volume Update Package
	SDK-3	Generator of update script running on Target Device
	SDK-4	Plug-in mechanism to adjust to package host server requirements Sequence from TUP generation to upload
	SDK-5	A set of documents to use the SDK

Feature List Items

Overall

Each detail of O-1, O-2, O-3, O-4, O-5, O-6 can be found in System Overall.

Trusted Update Package

Trusted Update Package (TUP) contains multiple update packages called Inner Packages, has information of each update package as metadata, can partially extract and deliver only necessary data and realize flexibility and tamper-resistance.

UP-1 Flexibility allowing to add data/metadata in any format (multiple and proprietary format packages can be contained)

As described in System Overall, there are various Update Target types such as data like images, videos and software like OS and applications. Install location and order varies depending on the types of Update Targets. Therefore, it is necessary for the TUP format to be able to contain any data and metadata of the data. In cases where Target Device has multiple Update Targets and there are dependencies among them, one package should be able to contain multiple update data in order to handle each update data simultaneously.

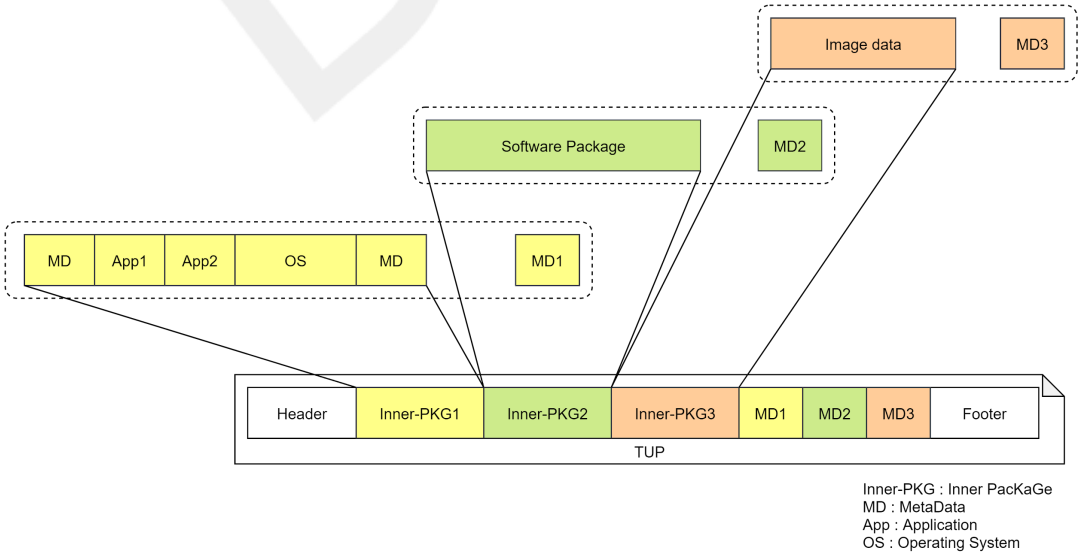

Fig. 14 Trusted Update Package(TUP)

UP-2 Free allocation of data/metadata

The optimal data configuration may differ depending on Domain Controller and Updater. Therefore, the allocation of data in the TUP is not fixed, but can be allocated in a data order and alignment that is easy to handle by Domain Controller or Updater, thereby increasing the efficiency of data handling.

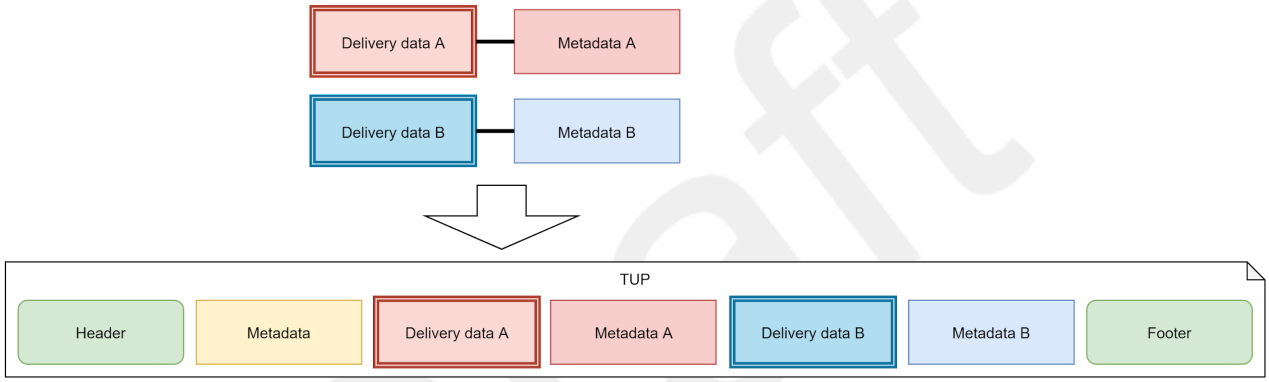

Fig. 15 TUP Allocation

In addition, as shown in Fig. 16, different allocation than Fig. 15 can be defined.

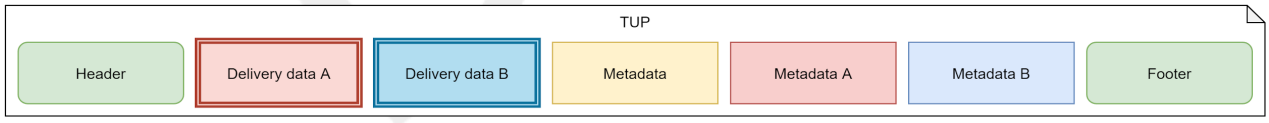

Fig. 16 The TUP Allocation Example

UP-3 Apply to devices with fewer resources (configuration for each device)

To allow devices with fewer resources to handle packages of the TUP, part of the processing (decryption and decompression) can be processed on another device, and the memory required for tampering check can be adjusted by changing the ICV (Integrity Check Value) block size written in metadata.

UP-4 Partial process without waiting for the whole data

It is possible to reduce the time required for the entire installation by downloading one or more parts of the necessary data at the same time instead of downloading the whole package, and then proceeding with the process sequentially once the necessary data is available.

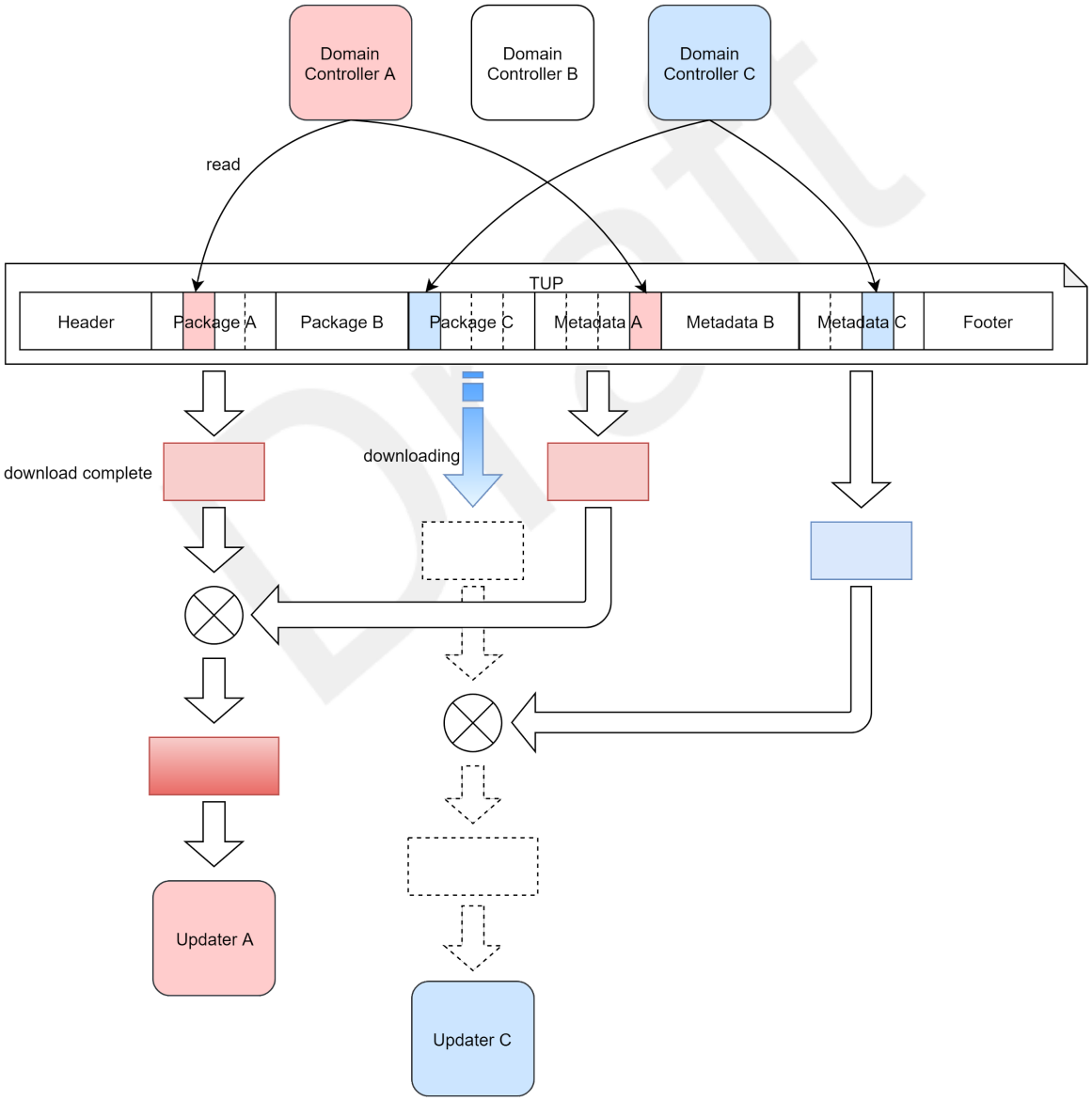

Fig. 17 Just in time download

UP-5 Structure to extract/deliver only necessary data

Even if all the data is eventually downloaded, the update process does not necessarily require all the data at the same time. Also, when updating multiple packages, there may be cases where some of the packages have already been updated, and where the user can choose which packages to update. The TUP may contain packages that are not actually used. In order to be able to handle these cases, we will implement a structure that allows users to download only the parts they need, rather than a format that only allows batch downloading.

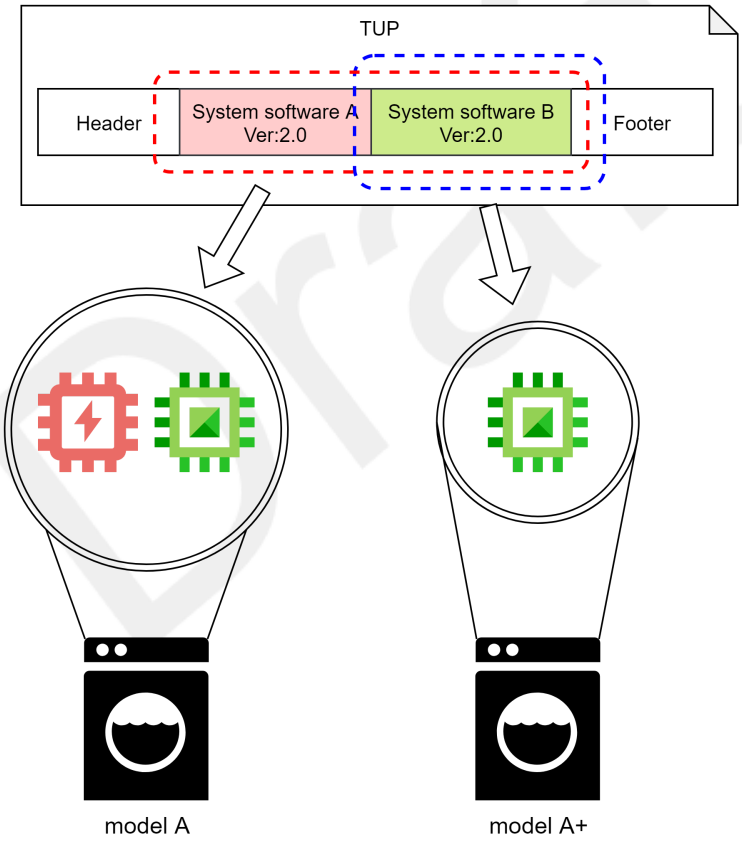

Fig. 18 Downloading only necessary part

The TUP can be used for multiple and different Target Devices that have common parts. For example, as described in Fig. 18, one TUP is used for both model A and model B. The model A uses both System software A and System software B TUP for update, but model A+ uses only System software B for update. This means that an additional TUP which contains only System software B for model A+ is not necessary to be created.

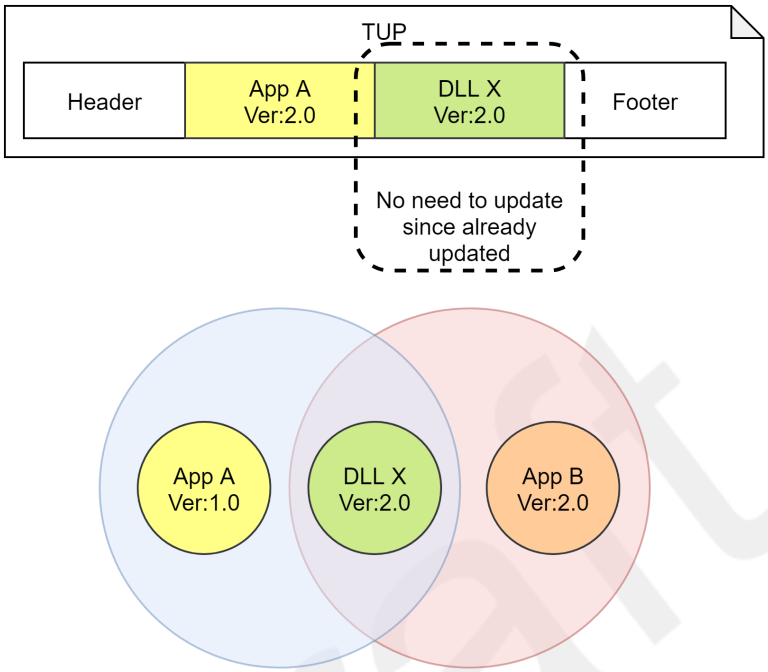

Fig. 19 Update Target strategy

In addition, in cases where some Update Targets' version already meets their version requirements, the unnecessary updates don't have to be performed. For example, in Fig. 19, there is no need to update DLL X, which is used by both App A and App B, along with App A ver:1.0 since it was already updated when App B was updated.

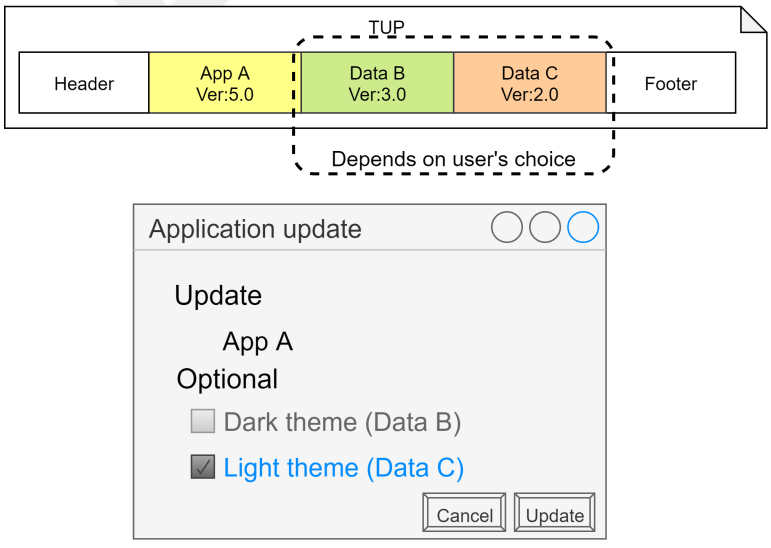

Fig. 20 Integration into UI system

In cases where a user can choose whether to actually update some data in a package, only the selected data can be used for the update. For example, in Fig. 20, Data B and Data C associated with App A can be updated or not, depending on the user's choice.

UP-6 Metadata format to realize forward/backward compatibility

Newer TUP Parser is able to interpret older format TUP, and there is backward compatibility that newer TUP Parser can appropriately behave in a way older format TUP instructs. With this backward compatibility, an existing TUP can be used without regenerating it even when a newer TUP Parser is used.

In addition, there is forward compatibility that an older TUP Parser can read and skip an unknown data type in TUP metadata since it is in TLV(Type Length Value) format. With this forward compatibility, when there is an update in TUP format, which is not mandatory, updating TUP Parser can be arbitrary.

UP-7 Tamper/damage check at every package read

If update packages are checked only when they are downloaded, like in Fig. 21, it is not possible to find a tamper/damage that occurs between download and actual use.

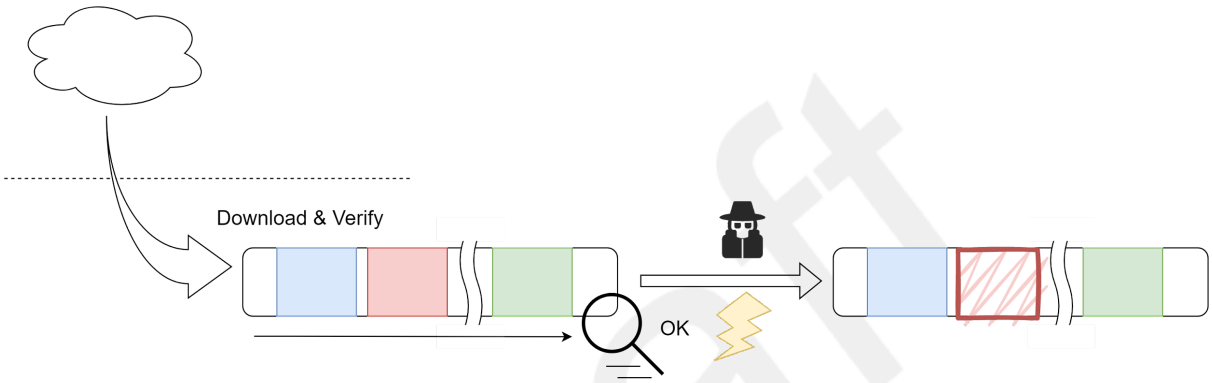

Fig. 21 Example of verify only when downloading

Therefore, the TUS checks data every time it is used. In order to realize this, as described in Fig. 22, the TUS allows not only a whole data check that may cost a lot but also a partial check that is performed for only the actually used part and can be done with low cost.

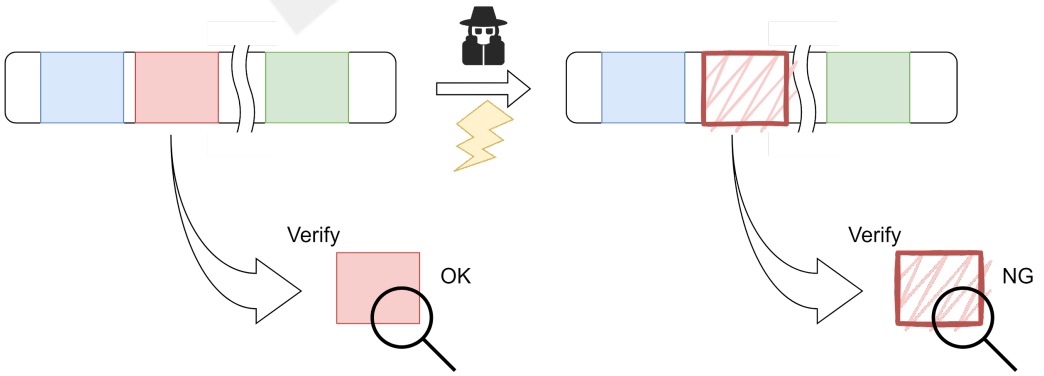

Fig. 22 Example of verify when using

UP-8 Tamper/damage check of only necessary part without a whole data

In cases where only checking of the whole data is supported, all the data in a package needs to be downloaded on a device. When the download size is not small enough to be loaded on memory, the data needs to be written on storage. In addition, after a whole data check is performed, it is necessary to guarantee that all the data has not been tampered until it is used. If the guarantee is not possible, a whole data check needs to be performed every time the data is used.

Every storage has its write cycles and every write on a storage consumes its write cycle. If the size of the partial check is small enough, it can be done only on memory and does not consume storage writes.

In order to check data partially, the data is divided into blocks of a certain size, and ICVs are calculated for each block. If the set of ICVs obtained is larger than the block size, ICVs are repeatedly calculated for each block to generate a hierarchical set of ICVs. If the Root ICV is verified once, it can be cached as having been verified up to the leaf.

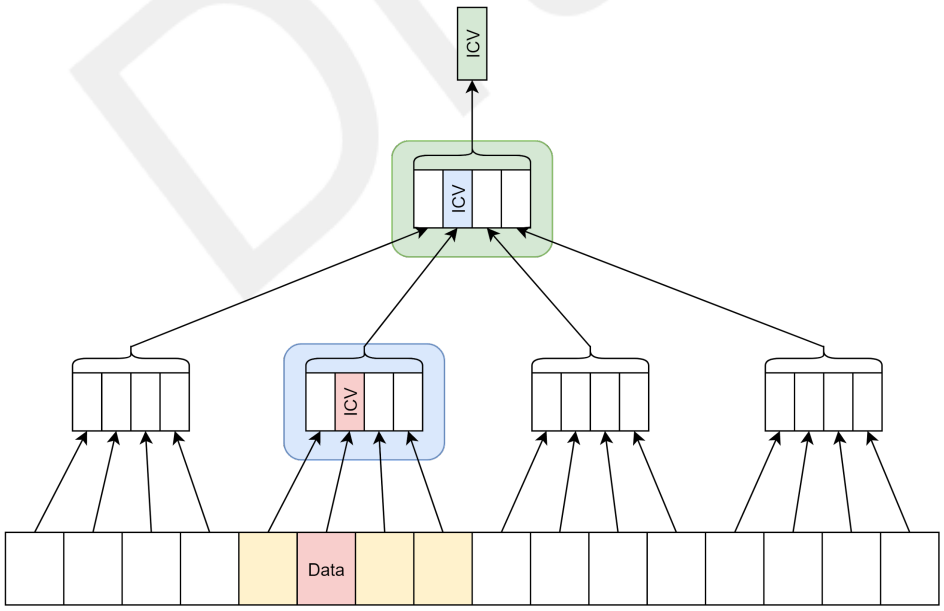

Fig. 23 ICV-Tree

UP-9 Encryption, sign and compression for each data range depending on requirements (Security requirements such as encryption strength and algorithms and device constraints)

Each Update Target might have different encryption and compression methods. Therefore, TUP needs to be able to handle encryption or compression methods that are suitable to each Update Target and apply them to each update package.

Developers can choose compression with or without random access, and if the latter one is chosen, the size of the metadata related to compression can be reduced. In addition, in cases where an Update Target does not have enough resources to decrypt or decompress, another device can handle the necessary process instead.

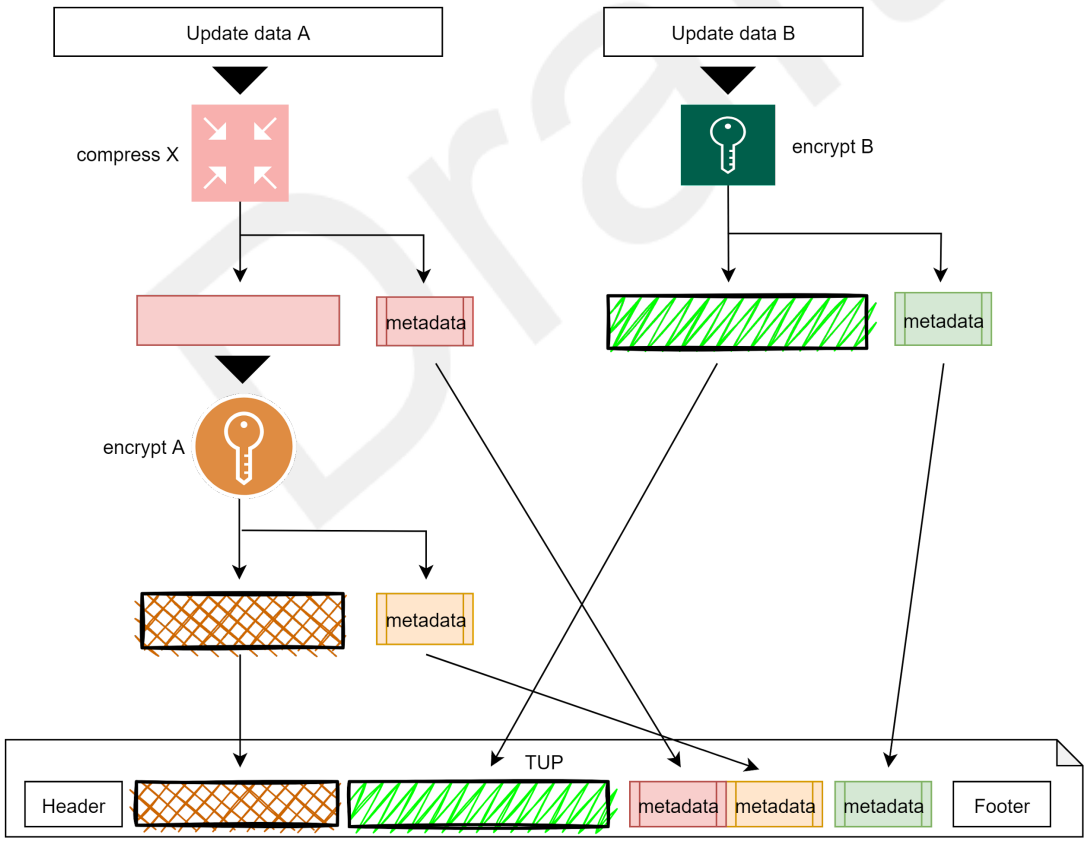

Fig. 24 Encryption, Sign and Compression

UP-10 Protecting and obfuscation of update content

By using a proprietary data format instead of an existing general data format, it is possible to prevent easy creation of packages and thus prevent the creation of packages whose generation method is unknown or that have been tampered with. In addition, TUP files and update data are encrypted and ICVs are added to them, but other processes can also be added in order to enhance tamper resistance.

Finally, the general flow to parse the TUP and retrieve the Inner Package is described below.

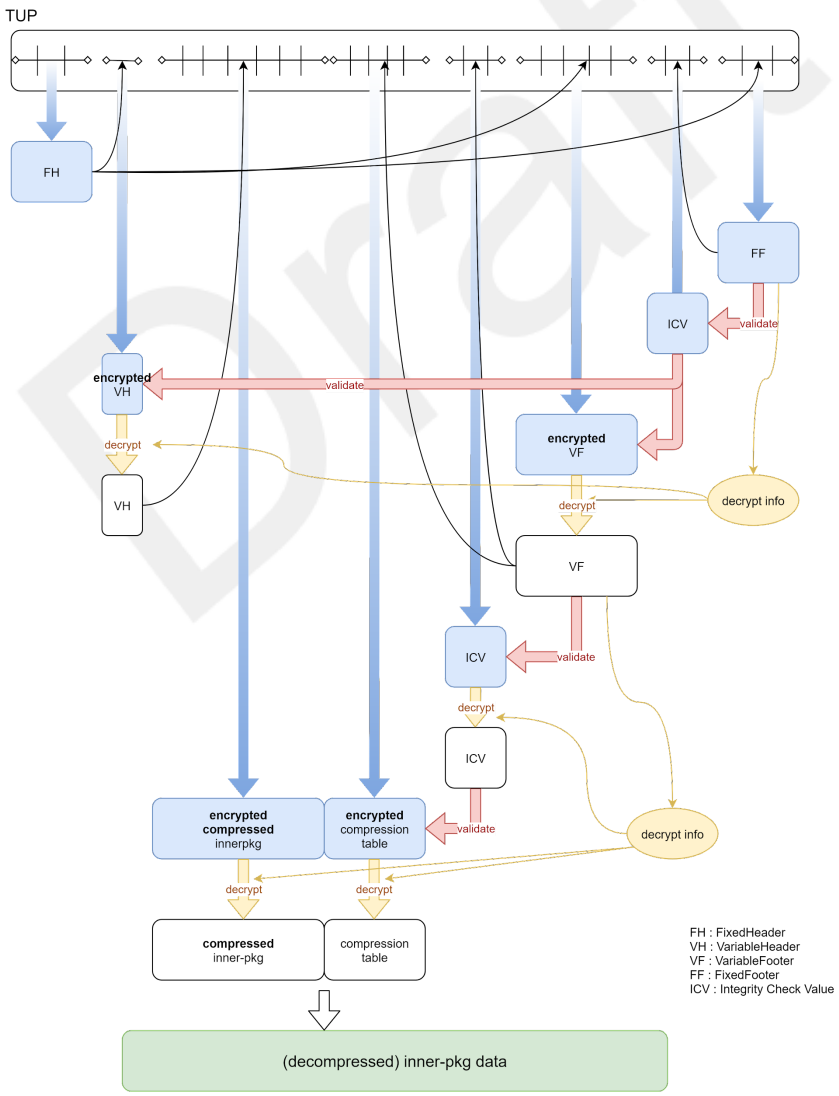

Fig. 25 Extracting Inner Package from TUP

Target Device

There are supposed to be very diverse types of Target Devices in terms of hardware specifications, network structure, system structure(single or distributed), platform environment like bare metal, a specific OS and middleware, etc.

The TUS provides distributed and easily extensible runtime of script in order to realize updates on any environment.

TD-1 Flexible update control using script

A Target Device with complex structure may need to update each system of the device one by one, which is dependent on each other. The update control might include a complex process such as a resume after another system's update and a reboot after the update. In order to complete (or cancel properly) update on such a complex Target Device without any error, as described in Fig. 26, the TUS provides script runtime on Target Devices and the TUP format which can contain the script. Update method logic can be written on the scripts and be executed on Target Devices.

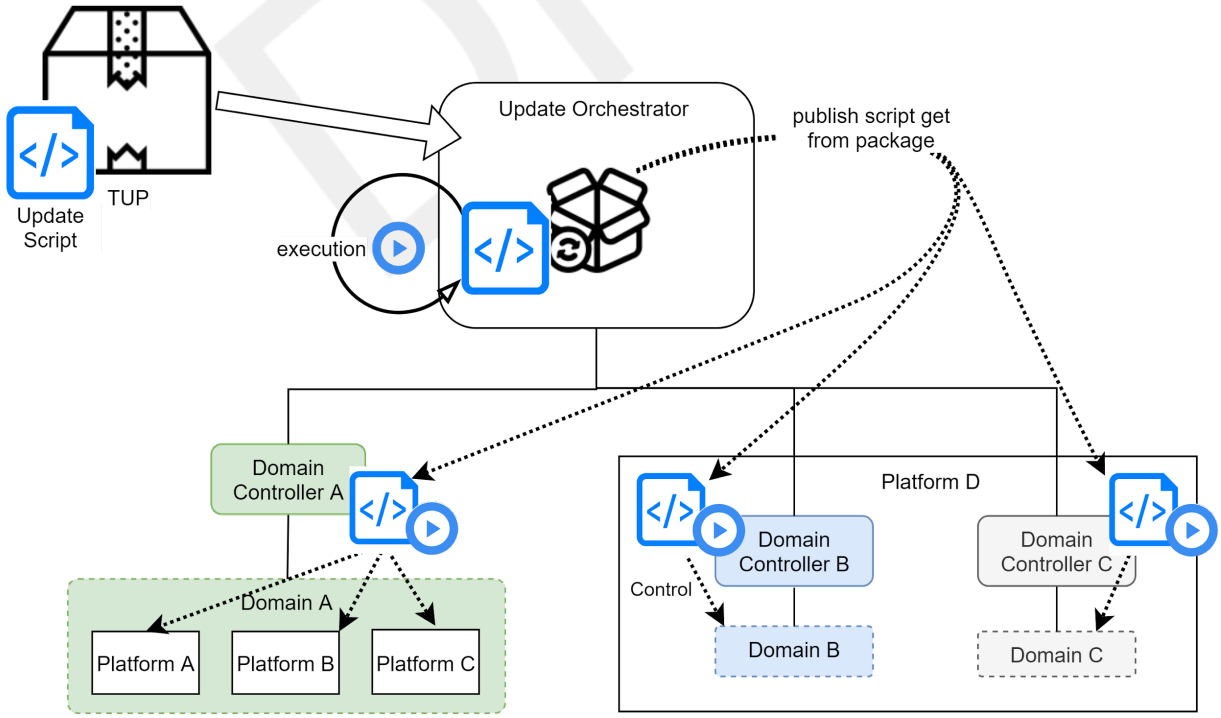

Fig. 26 Update orchestration with scripting environment

The TUS provides primitive APIs for updating processes with scripts. A simple system can be updated by a simple script using these APIs. In addition, since the script runtime is easy to add functions (TD-2), it is possible to describe update control that can be applied to various environments.

There is also a mechanism that provides developers to prepare the script, in order not to write it manually (SDK-3).

TD-2 Abstraction layer to reduce porting cost

As shown in Fig. 27, the TUS provides Platform Abstraction Layer (PAL) for Update Orchestrator, Domain Controller and Updater running on Target Devices. Platform Abstraction Layer abstracts API functions of each platform such as storage I/O and network I/O and also provides functions such as encryption/decryption and compression/decompression, as described in Fig. 28. The TUS provides a reference platform and its reference implementation. Developers can easily implement porting to another platform and also implement additional functions specific to another platform.

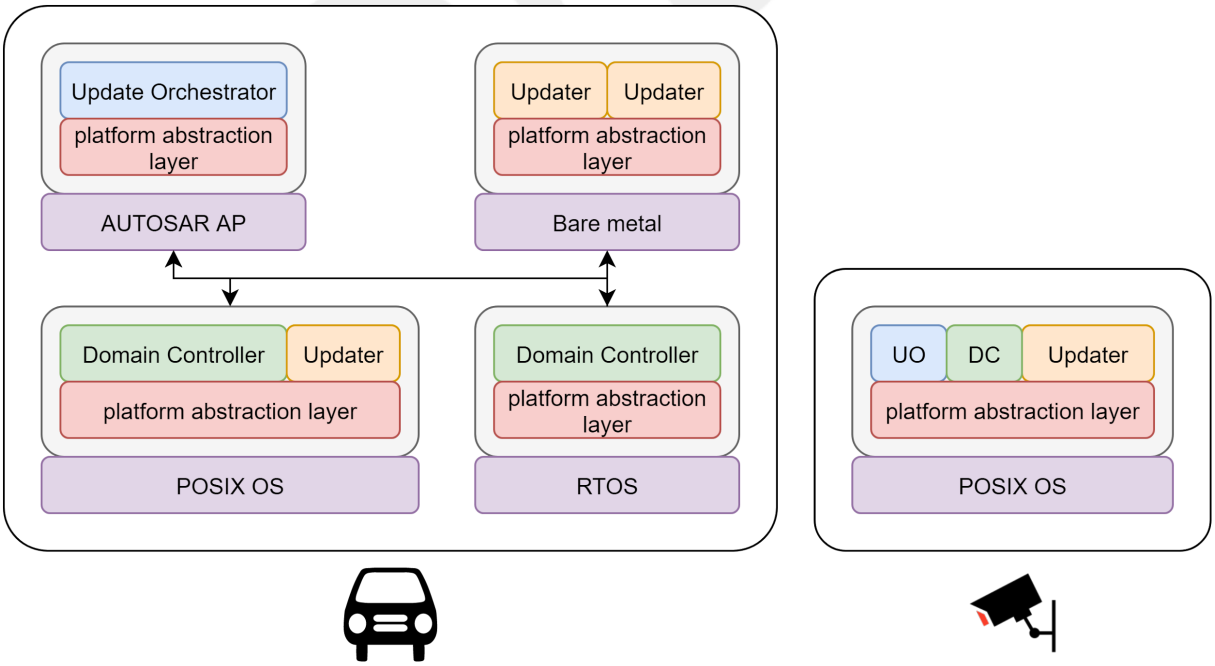

Fig. 27 Platform Arbitration Layer

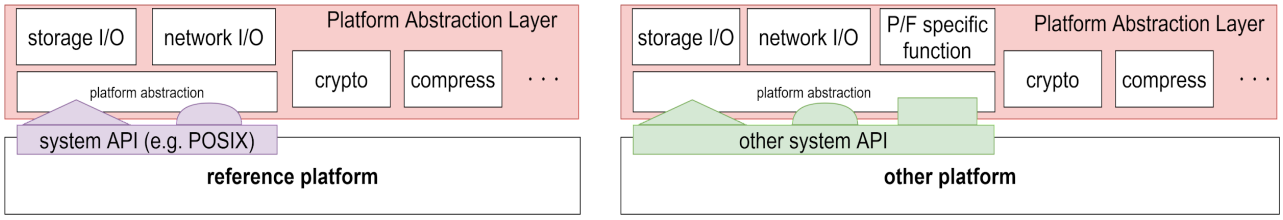

Fig. 28 Examples of reference platform and vendor platform

TD-3,4 Using hardware accelerator,
/ High speed process on Target Devices with sufficient resources

Some functions of PAL such as encryption/decryption and compression/decompression might be implemented to use hardware accelerators. Therefore, the TUS's PAL provides an API layer whose each function can be replaced with hardware vendor's implementation, and developers can effectively use hardware implementation provided by them. This mechanism also contributes to speeding up the update process with Target Devices' resources as much as possible.

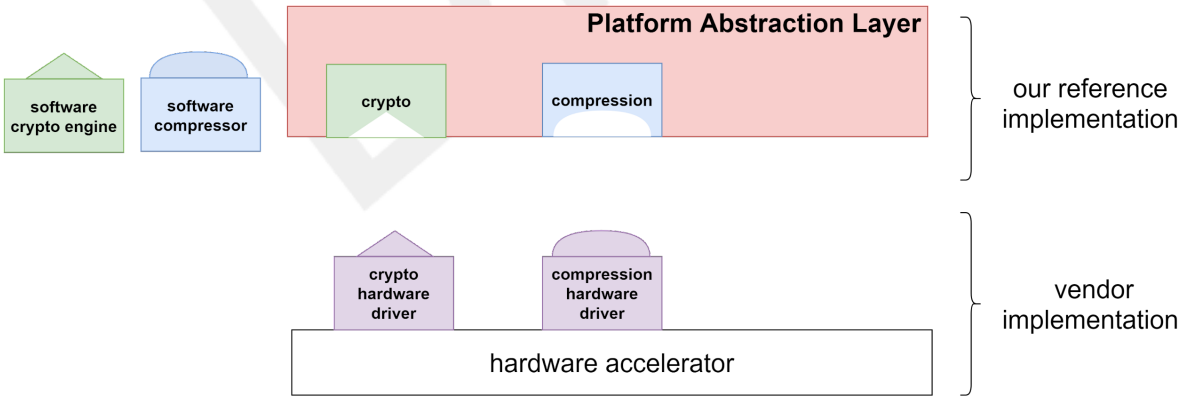

Fig. 29 Abstract hardware accelerator and system platform

TD-5 Using hardware security functions

In cases a Target Device has a function to divide execution permission as a hardware, each software component can be allocated at an appropriate permission level so that important information such as encryption/signing key can be protected properly, as shown in Fig. 30.

For example, on a system which has 2 modes, user mode and privileged mode, Update Orchestrator, Domain Controller and Updater would be allocated at user mode. However, as described in Fig. 30, on a system which has some more modes with stronger permission levels, such as hypervisor mode or secure mode, each component can be allocated at each appropriate mode.

In addition, as shown in Fig. 31, some systems don't have privileged mode. Update Orchestrator and Domain Controller basically should not run on such a system, but Updater that receives commands from them can be allocated.

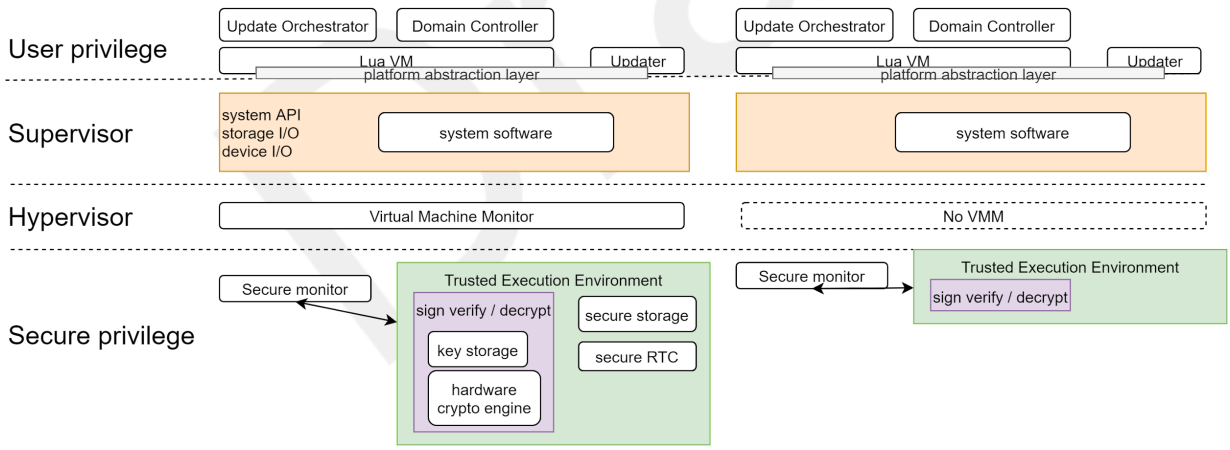

Fig. 30 Using security function of hardware

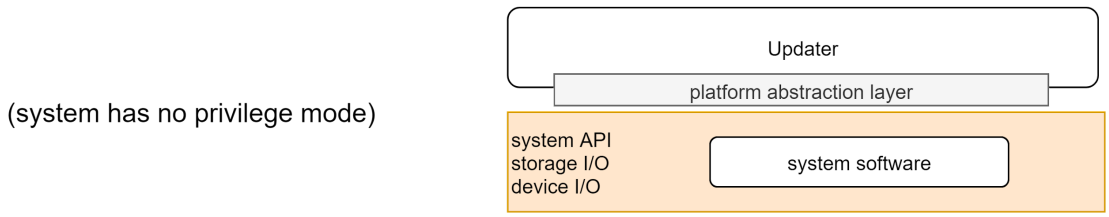

Fig. 31 Privilege control

TD-6 Integration with external systems (UI/Target status management server)

Target Device must have one Update Orchestrator to control the whole update of Target Device and it includes functions such as downloader and authenticator. As shown in Fig. 32, any communication protocols between OTA Center and Updater Orchestrator can be defined either wired or unwired and not only OTA Center but also other components such as maintenance tools can be used.

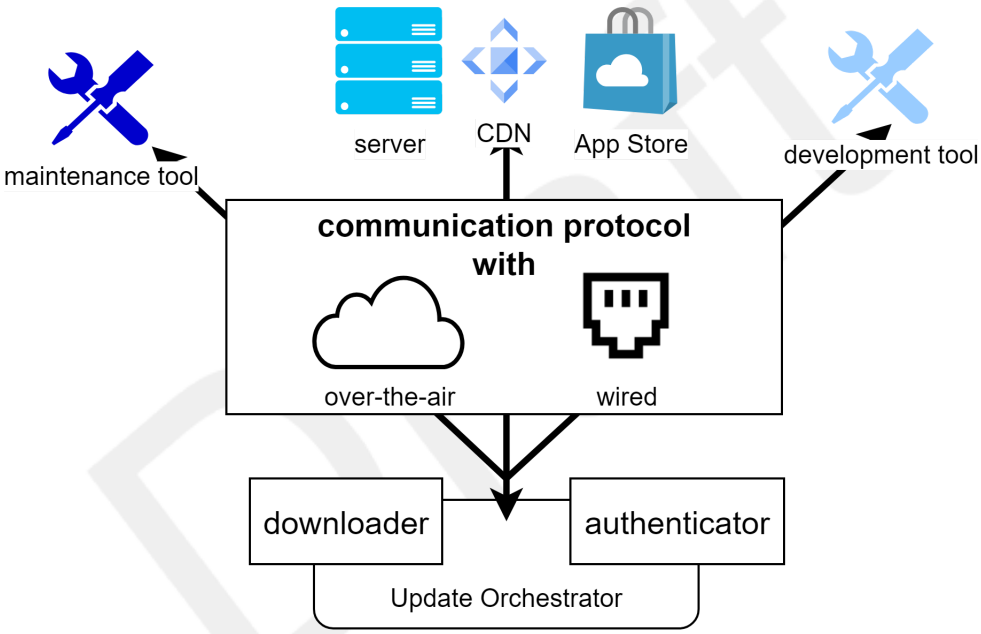

Fig. 32 Communication between upstream and TUS runtime

Update Orchestrator also has APIs to integrate with external modules such as GUI, update status management server and debug logging tools. Therefore, it can notify the update's status and interact with the user of Target Device, as shown in Fig. 33.

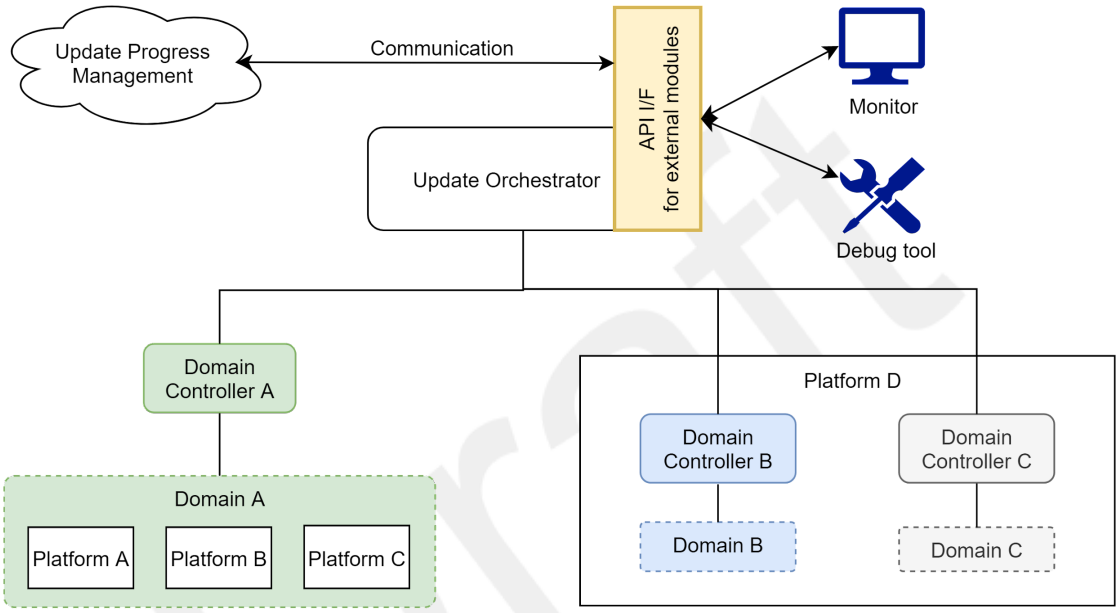

Fig. 33 TUS communicate with external modules

SDK

The TUS is provided in the form of SDK with developers and it includes below items.

- A set of tools such as TUP generator and parser
- Software components to update
    - Update Orchestrator
    - Domain Controller
    - Updater
    - Reference implementation under Porting Layer on reference platform
- Samples to try update on reference platform
- Development documents such as API Document and Getting Started

SDK-1 A set of tools to apply to various development flows and environments

A set of tools developers can use during the development is called Host-Tools and its purpose is the following.

Adjust to any development environments

In general, the best development environments vary depending on a lot of elements such as development phases or development strategies. Package management method for updates also varies depending on environments, thus keys for package signing also changes.

The TUS SDK provides Host-Tools that can be adjusted and used at any development environments. In addition, tools such as package generation tools with high efficiency and emulators like SILS are also provided in order to contribute to enhance development efficiency.

The minimum list of the Host-Tool of the SDK is the following. In addition to this, tools like analyzer for debug also would be provided. Relationships between each tool are described below.

Table. 2 Host-Tool List

Name	Abstraction
Publishing Tool	・Includes TUP generator, script generator of update methods and uploader to upload generated packages to specified hosts.
TUP Parser Library	・Library to analyze TUP. ・Can be used to verify TUP if necessary.

SILS	・Software emulation environment that can test updates without Target Device. ・Consists of components Update Orchestrator, Domain Controller and Updater in order to test updates.
Dummy OTA Center	・Minimum host server of packages to be used during development. ・Has functions of at least TUP upload receiver, TUP host and download responder.
Deployment Tool	・Tools to deploy TUP from a development machine to Target Devices directly. ・Contributes to enhance development efficiency by bypassing the Center at early development phases.

In addition, as in Fig. 34, Host-Tool provides tools to adjust to various development phases.

For example, TUP generated by Publishing Tool can be :

- deployed directly to SILS//Reference Target (Ref Target).
- deployed to SILS/Ref Target after uploading to Dummy OTA Center.

Pipeline like the above can be selected flexibly depending on development phases.

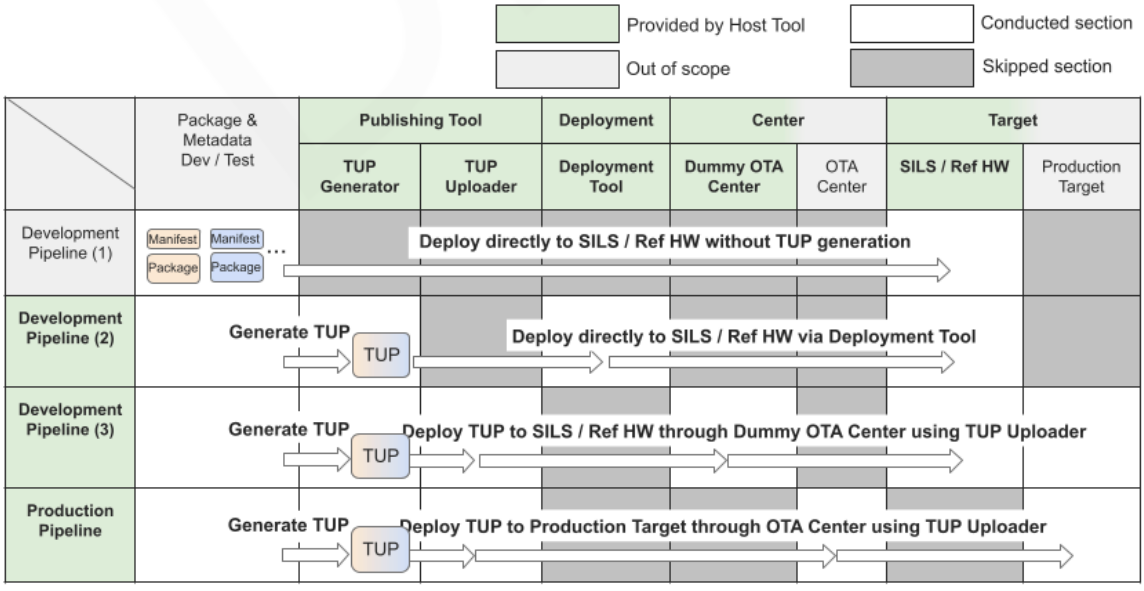

Fig. 34 Relation between development pipeline and Host-Tool

SDK-2 High speed generation of large volume Update Package

Since TUP generation is embedded into development flow, high speed generation contributes to :

- enhance development efficiency
- quick update of a product as much as possible (e.g. emergency update for found vulnerability)

Fig. 35 describes an example of the high speed generation process of TUP. In this example, an Inner Package is required to be compressed when TUP is generated and it should be compressed, encrypted and signed in this order, per fixed sized block. Since the necessary size of a block is determined upfront for each process, the proceeding process does not need to wait and the next process can be started when it is ready.

In Fig. 35,

- Compression is started for each fixed sized block of No. 1 - 8. Block No. 5, 6 and 8 are still under calculation since the compression process would not always be completed in order from the beginning.
- Encryption is started at the moment when the compressed block size reaches the size of the encryption block. In this case, block No. 1, 2, 3 and part of block No. 4 are encrypted and ENC. A and ENC. B are generated.
- Signing is started at the moment the encrypted block size reaches the size of the signing block. Though it is not described in the figure in detail, processes such as ICV Tree generation and adding it to TUP as metadata will be done afterwards.

High speed TUP generation is possible by processing asynchronously like the above.

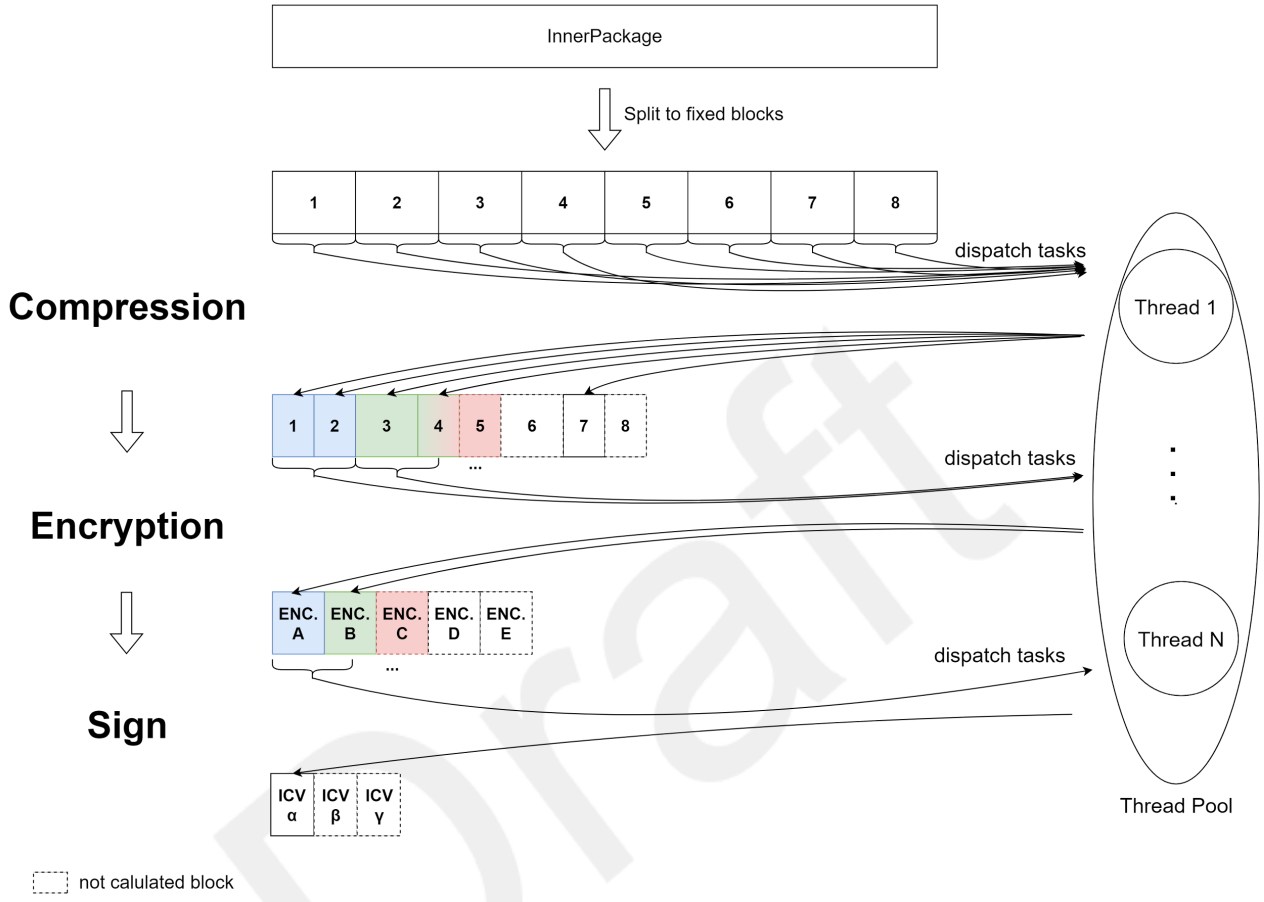

Fig. 35 Concurrent TUP generation

SDK-3 Generator of update script running on Target Device

As described in TD-1, scripts are used to control updates on Target Devices. Implementing these scripts manually is a burden for developers and might cause incidents due to mistakes in script implementation. Therefore, as one of the Host-Tools, the generator of the script is provided in the SDK.

Fig. 36 shows an overview of this generator. It generates a script with an inputted manifest which is common to all the Target Devices and a setting information of device-specific, and then generates a TUP for the Target Device including the script.

As described later in Fig. 37, the TUS allows extending Updater specific features. In this case, auto-generation of the scripts might be difficult. Therefore, not a generator but a linter and template

generator may be provided (T.B.D). In such a case, if update conditions are not met with necessary Setting information and Manifest (e.g. App B version required by App A is not appropriate), error would occur.

Implementation language of the update script would be Lua since it has features like the following.

- Footprint of the runtime environment is small and execution speed is good among script languages.
- Proven track record of adoption in products such as game hardware and network routers with severe resource constraints.
- No need to care about CPU architecture since it runs on a VM.

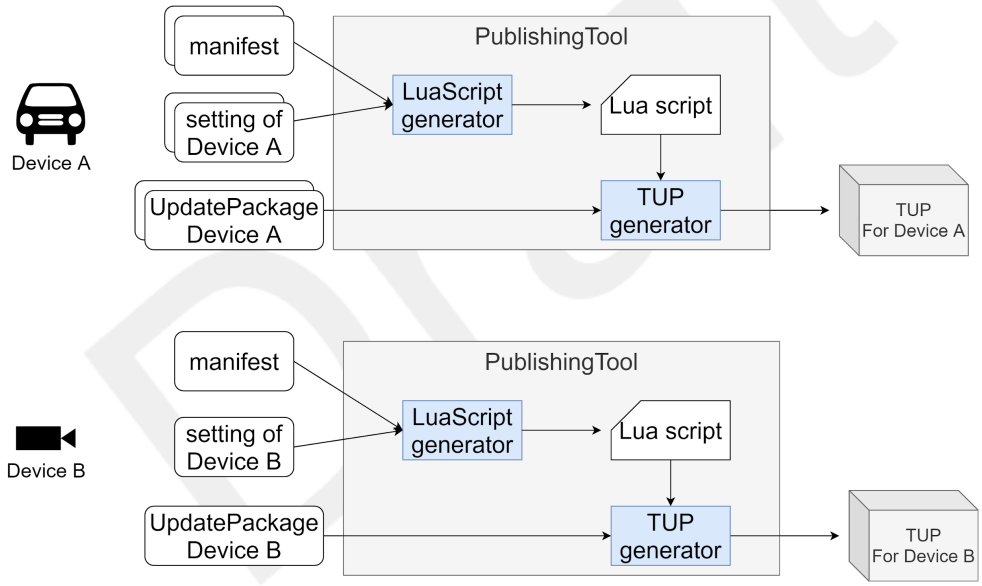

Fig. 36 Lua Script Generator

In addition, as mentioned earlier, the TUS has an extensibility to support specific functions of Updaters on Target Devices.

Since SDK users can add their own Lua Extensions to Domain Controller, which interprets Lua scripts and controls the Updater, the script generator must be able to generate Lua scripts that call the functions of the extensions added by the user.

As shown in Fig. 37, a format to describe the extension features added by the user would be defined, and the generator would interpret it and generate a Lua script containing those. The

generated Lua script would be packaged into the TUP and delivered, thus updates specific to a certain Updater would be realized.

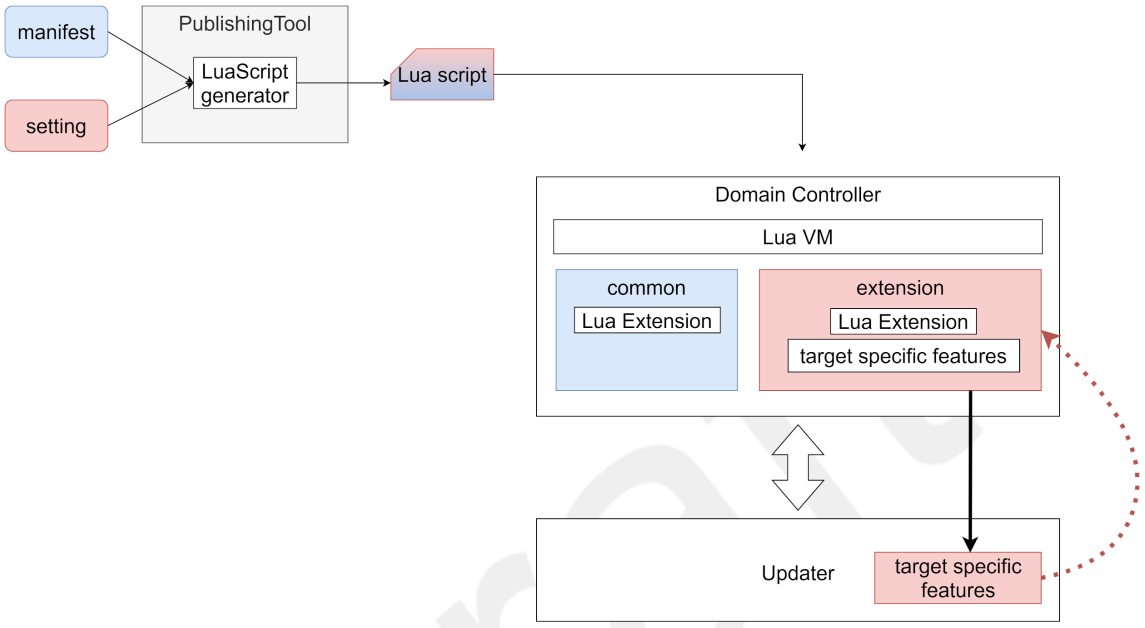

Fig. 37 Extension Lua Script Generator

SDK-4 Plug-in mechanism to adjust to package host server requirements

In general, environments of the TUP host server to upload and associated keys vary depending on development environments. In order to deal with this difference, the Package Uploader included in the Publishing Tool has a plug-in mechanism that allows the upload function to be exchanged depending on the environment in which it is provided.

Fg. 38 shows TUP generation flow and deployment.

The Publishing Tool generates a TUP using the package created by the developer/operator and the associated manifest as input. The generated TUPs can be uploaded to the Dummy OTA Center / OTA Center or deployed directly to SILS / Ref Target through the Deployment Tool depending on the usage.

As mentioned above, the Uploader included in the Publishing Tool has a plug-in mechanism, and it can be exchanged depending on the Center environment. The APIs and reference implementations of the plug-ins used in this case are provided in the SDK, and the respective developers of the TMC or other OEMs Center provide plug-in implementations corresponding to each environment.

For example, if some TUPs need to be uploaded to the OTA Center, which is a production environment managed by a certain OEM, the production operator of the target OEM can create a plug-in of the Publishing Tool for the SDK users. Once the TUP is uploaded to the Center, the TUP Parser Library provided by the SDK can be used to verify the TUP at the Center if necessary.

In addition, when a TUP is passed to the Deployment Tool, the TUP can be deployed directly to SILS / Ref Target by either wireless or wired method.

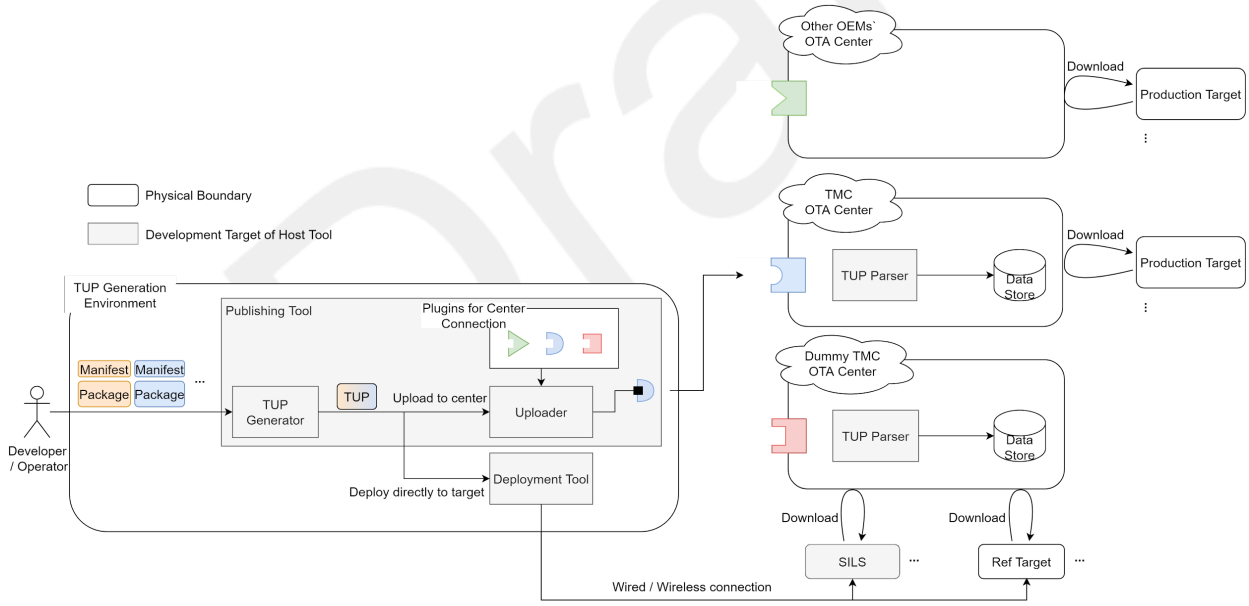

Fig. 38 Host-Tool component diagram

Sequence from TUP generation to upload

Fig. 39 describes the abstract flow of TUP generation and upload using the Publishing Tool.

The Publishing Tool consists of TUP Generator, Lua script Generator and Uploader to upload to a specified host if necessary. The Publishing Tool takes multiple packages and Manifest as inputs to generate TUP, then uploads the generated TUP file to the hosting Center if necessary. TUP generation includes generating Lua scripts which describe the update method and also includes processes such as compression, encryption and signing of the TUP file depending on the necessity.

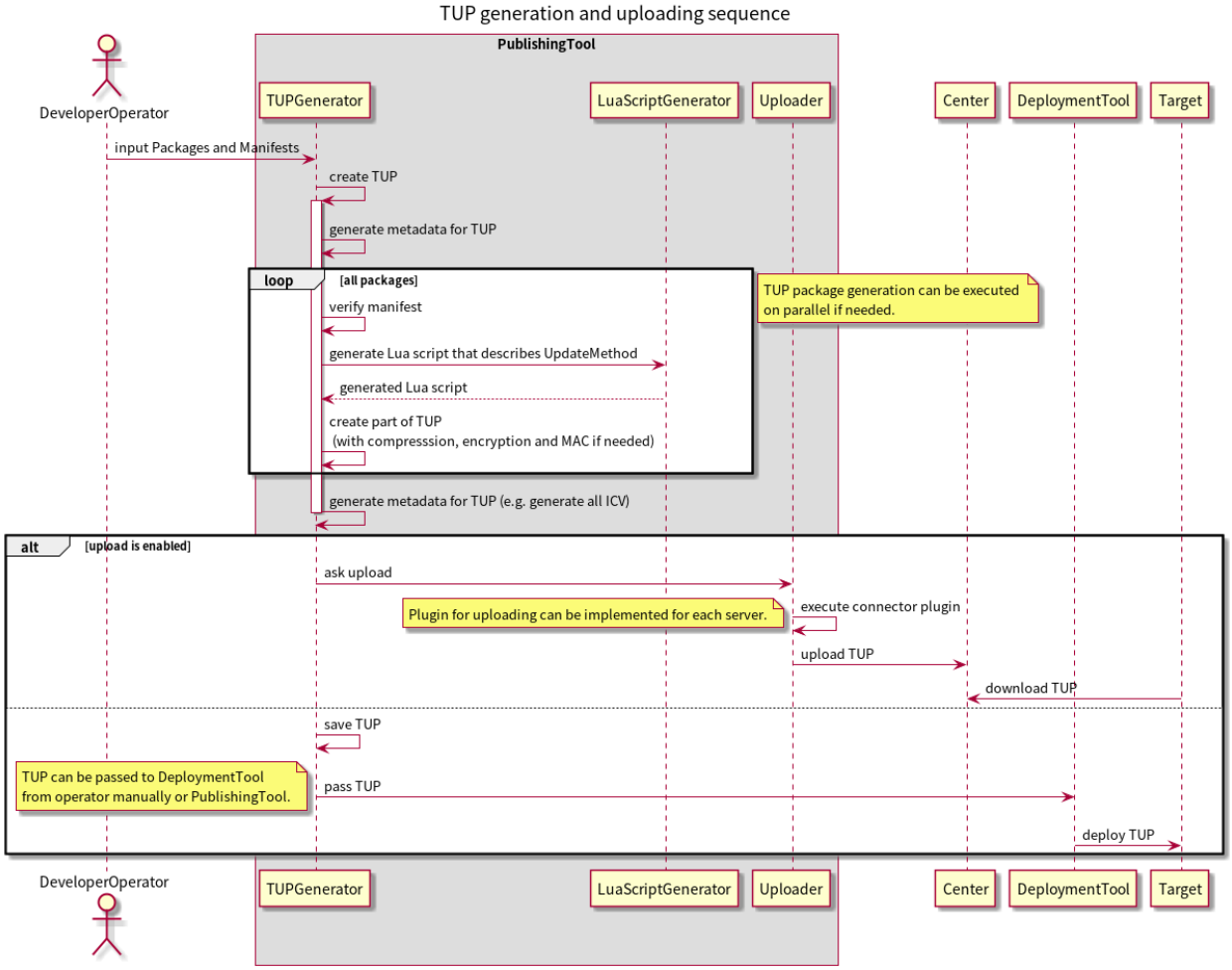

Fig. 39 TUP generation and uploading sequence

SDK-5 A set of documents to use the SDK

The SDK contains documents necessary for the development using the SDK.
(e.g. Release note, API Documents, Porting Documents, Getting Started for SILS/Ref Target environment, etc.)

Quality Criteria

This section describes a list of quality criterias that the TUS will meet. Each quality criterias will be achieved by the TUS features mentioned above.

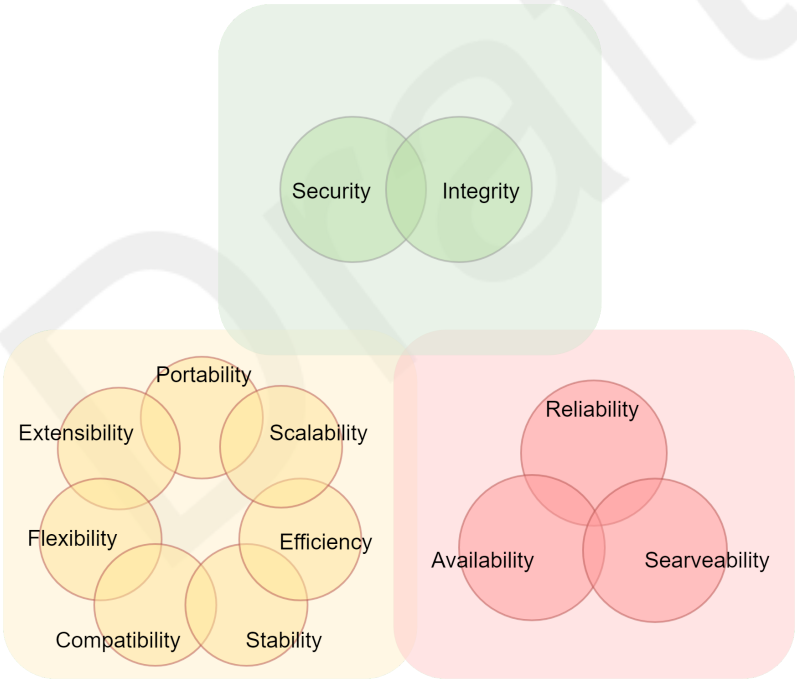

Fig. 40 Quality Criteria Relationships

Table. 3 Quality Criteria List

Quality Criteria	Related Features	Description
Security	UP-7, UP-8, UP-9, UP-10, TD-5, SDK-3	It is essential to provide security assurance measures through encryption and signature depending on the risk rank of the target system and the function of the target hardware.
Integrity	UP-7, UP-8	Even if the Update Package is from a trusted source, it may have been tampered or damaged along the network path. Regardless of the encryption method, it is necessary to be able to confirm the integrity of the TUP on the target device to ensure that these risks do not exist.
Portability	O-2, UP-1, UP-2, UP-3, TD-2, TD-3	The target device may be a one with limited resources. It needs to be as configurable and as small a footprint as possible so that it can be applied to any devices which need to be updated.
Extensibility	O-1, O-2, O-3, O-4, O-6, UP-1, UP-6, TD-1, SDK-3, SDK-4	In the future, it is assumed that the system configuration of the Update Target and update requirements will change significantly. It is necessary for the TUS to have the ability to cope with such future changes.
Flexibility	UP-1, UP-2, UP-3, UP-8, TD-1	Even for packages with the same purpose, the user/developer needs to be able to flexibly change the contents in consideration of performance, resource consumption and generation time.
Compatibility	O-2, UP-6	Since devices with various versions of the TUS system might be co-existed and used in the market, it is necessary to ensure forward/backward compatibility of the system itself.
Stability	O-2, O-3, UP-6	There are a lot of variations of Target Devices of the TUS such as hardware specifications, OS and middleware. It is necessary for the TUS to be able to run stable on any Target Devices.
Efficiency	O-2, O-3, O-5, UP-2, UP-4, UP-5, UP-9, TD-3, TD-4, SDK-1, SDK-2, SDK-3	Package generation time in the development environment, update time in the device and network cost should be minimized as much as possible.
Scalability	O-1, O-2, O-3, O-4, UP-1, UP-3, UP-8, TD-2	The TUS should be applicable even if the type of Target Device or its internal structure changes significantly, and the service scale should be scalable depending on the necessity.
Reliability	O-2, O-3, TD-1	It is necessary for the TUS to have a mechanism to prevent system shutdown or malfunction of the Target Device due to

		update failure as much as possible.
Availability	O-2, TD-3, TD-4, TD-5, TD-6	For both general users and business operations, it is necessary to minimize the downtime of the Target Device due to the update process as much as possible.
Serviceability	SDK-1, SDK-4, SDK-5	In the event of an incident, the development and operation teams must be able to analyze the root cause and take measures to recover as soon as possible.