Trusted Update Package Format Specification

System Software Development Department,

InfoTech,

Connected Advanced Development Division,

Toyota Motor Corporation

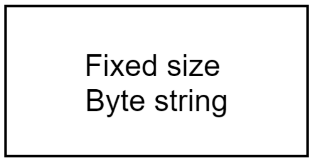
Contact : Okino, Naoto (naoto.okino@toyota-tokyo.tech)

Revision History

Version	Date	Description
1.0	20/10/2021	Initial release
2.0	5/9/2022	Add ned TLV(DELTA-PATCH, ICV-AFTER-DATA, ICV-BLOCK)

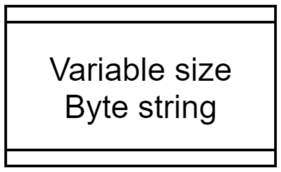
Data Format Types

Fixed size byte string



This represents a byte string whose length is fixed and has more than 1 byte.

Variable size byte string format



This represents a byte string whose length is variable and has more than 0 byte.

Metadata format

Metadata ID format

ID format (2 bytes) to identify types of metadata.

bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	N	R	R	R	T	T	T	T	T	T	T	T	V	V	V	X

N : Namespace (1 bit)
R : Reserved (3 bit)
T : Type (8 bit)
V : Version (3bit)
X : Type-dependent bit (The usage is determined by the Type)

ID Types

Table. ID Type groups

Upper 3 bits of ID Type value	Group
0x00	Data type
0x20~0xC0	Reserved
0xE0	Data format

Table. ID Type

Type value	Description
0x00	Invalid
0x01	Index information
0x02	Inner Package location in TUP
0x03	ICV-TREE information
0x04	ICV-ARRAY information
0x05	Inner Package information
0x06	Inner Package name
0x07	Compression (whole) information
0x08	Compression (fixed length block) information
0x09	Compression (variable block) information
0x0A	Encryption information
0x0B	Update flow
0x0C	Inner Package version
0x0D	Domain of Inner Package
0x0E	Processing order of the target data (compression, encryption and ICV)

0x0F	Delta patch
0x10	ICV of patched data
0x11	ICV-BLOCK information
0xFA	TLV format
0xFB	VariableFooter (TLV format)
0xFC	VariableHeader (TLV format)
0xFD	FixedFooter (Fixed order)
0xFE	VariableHeader (Fixed order)
0xFF	FixedHeader (Fixed order)

TLV (Type-Length-Value or Tag-Length-Value) format

TLV consists of three elements, which are Type, Length and Value.

Type	Length	Value

The Type represents the type of data in the Value area and the Length represents the size of the Value. TLV format data can be nested by containing TLV data in the Value. In addition, even if a Type value is unknown, by reading its Length and skipping its Value, accessing the next TLV data is possible.

TLV format (Version 1)

For the details of Type, refer to Metadata ID format. The X bit in Metadata ID format is interpreted as padding flag P.

bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	N	R	R	R	T	T	T	T	T	T	T	T	V	V	V	P

N : Namespace (1 bit)

R : Reserved (3 bit)
T : Type (8 bit)
V : Version (3 bit)
P : With padding (1) / Without padding (0)

TLV format without padding

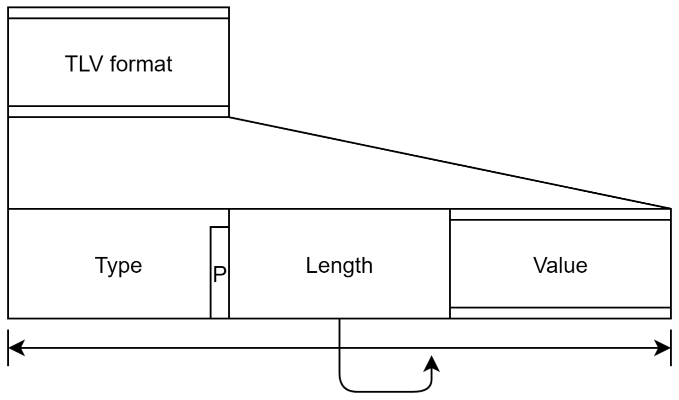


Table. Data structure of TLV format without padding

Name	Size (byte)	Description
Type	2	Data type in the Value
Type(P)	0（Included in Type）	1 bit to represent existence of padding in the Type (Always 0)
Length	6	Byte size of the length of the whole TLV
Value	Variable length (Length - 8)	Stored data

TLV format with padding

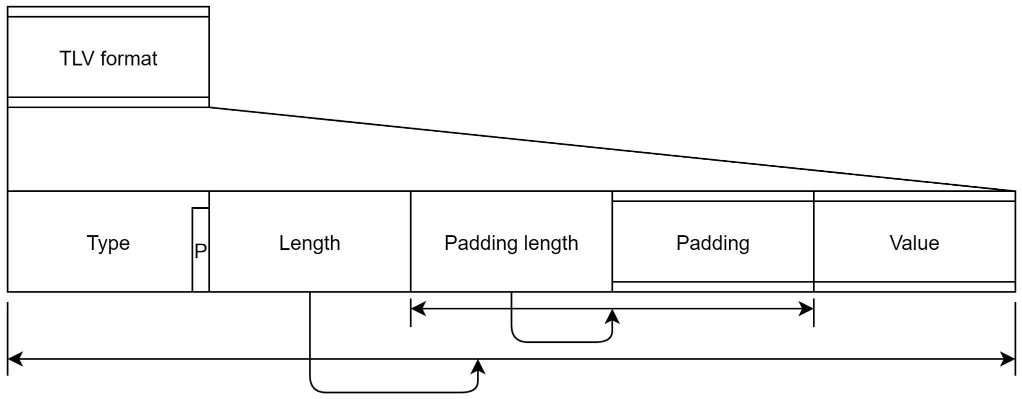


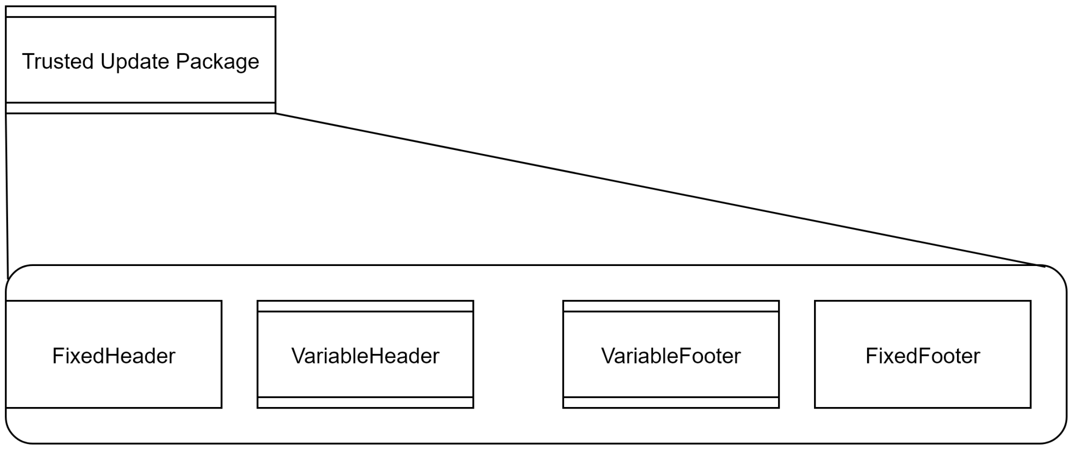
Table. Data structure of TLV format with padding

Name	Size (byte)	Description
Type	2	Data type in the Value
Type(P)	0（Included in Type）	1 bit to represent existence of padding in the Type (Always 1)
Length	6	Byte size of the length of the whole TLV
Padding Length	4	Byte size of the length of padding including itself
Padding	Variable length (Length - 4)	Padding to adjust alignment of the Value
Value	Variable length (Length - 8)	Stored data

Endian

The Version in the FixedHeader represents whether the byte order of the TUP file is Big Endian or Little Endian.

Trusted Update Package (TUP)



The Trusted Update Package (TUP) consists of four blocks, which are FixedHeader, VariableHeader, VariableFooter and FixedFooter. The FixedHeader is always located at the first part of the TUP. The location information of the other blocks are stored in the FIxedHeader.

FixedHeader (FH)

The data structure of the FH is determined by the value of FH type. However, the first 12 bytes always must be FH type and Version. Currently, only the Fixed Order can be used for FH type.

FH type : Fixed Order (Version1)

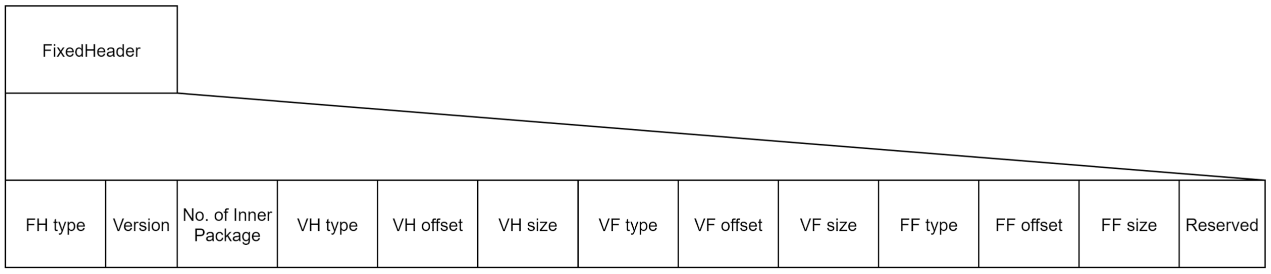


Table. FH Fixed Order (Version1) data structure

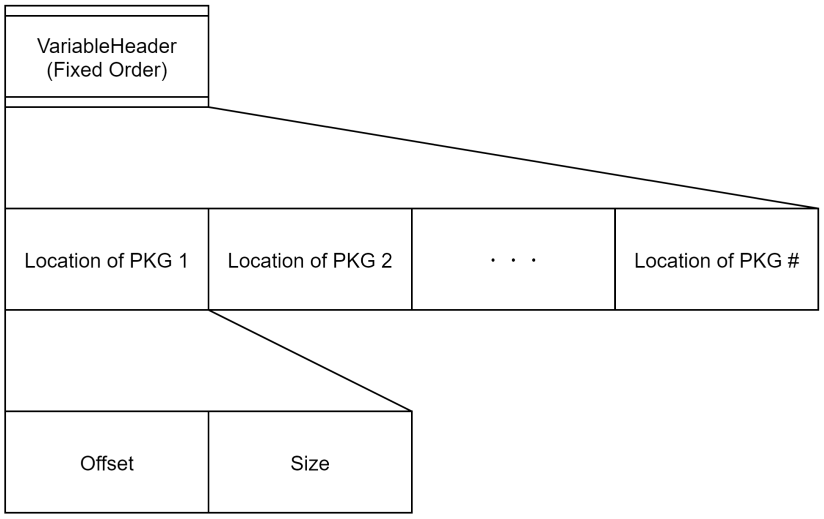
Name	Size (byte)	Description
FH type	8	Represents data structure of FH. See the bottom of this table for the format.
Version	4	TUP’s identification information. It is also used to identify endianness, so byte swapping will not result in the same value.
The number of Inner Package	4	The number of Inner Packages in TUP.
VH (VariableHeader) type	8	Represents data structure of VH. See the bottom of this table for the format.
VH (VariableHeader) offset	8	Offset from top of TUP to VH.
VH (VariableHeader) size	8	VH block size.
VF (VariableFooter) type	8	Represents data structure of VF. See the bottom of this table for the format.
VF (VariableFooter) offset	8	Offset from top of TUP to VF.
VF (VariableFooter) size	8	VF block size.
FF (FixedFooter) type	8	Represents data structure of FF. See the bottom of this table for the format.
FF (FixedFooter) offset	8	Offset from top of TUP to FF.
FF (FixedFooter) size	8	FH block size.

VariableHeader (VH)

Data structure of VH is determined by the VH type in the FH.

- VH type
  - Fixed order
  - TLV format

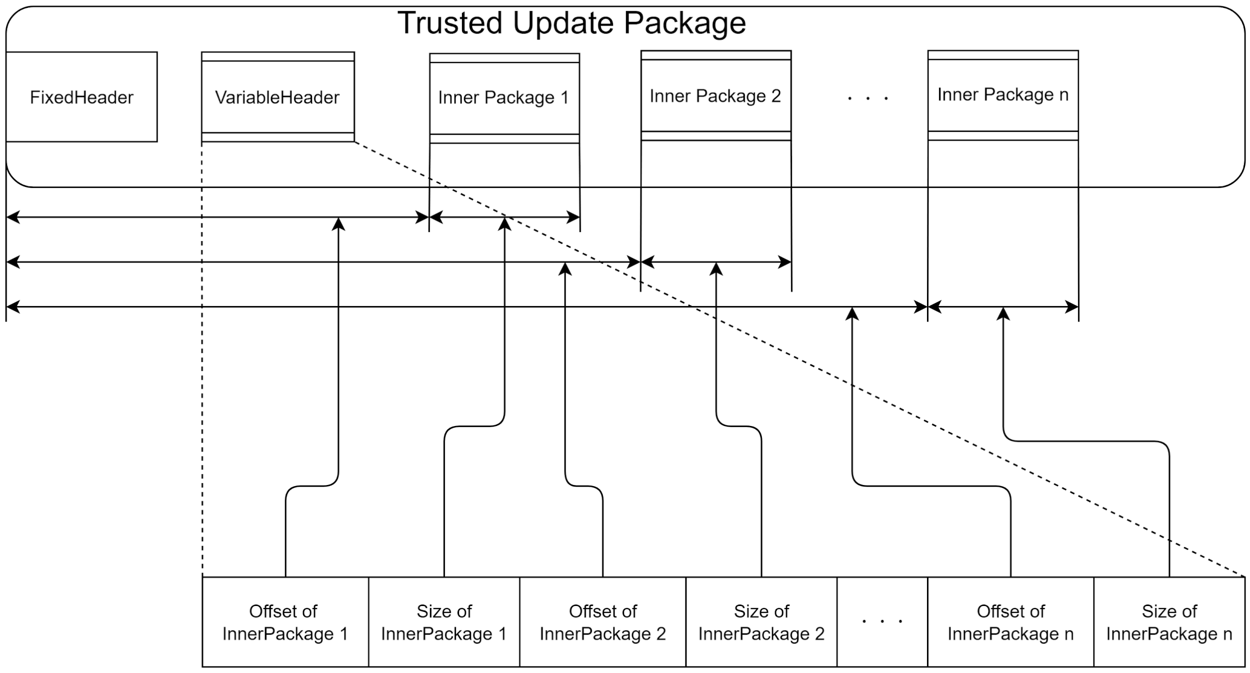
VH type : Fixed Order (Version1)



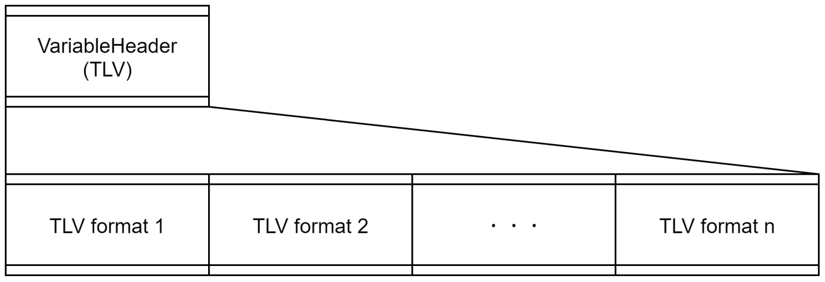
This format contains location information of each Inner Package. Each location information of Inner Packages contains the offset and the size. The total size of location information of every Inner Packages will be less than the size of VH. The order of Inner Packages information of Variable Footer is the same as the order of VH Fixed Order.

Table. Location information of Inner Packages in VH

Name	Size (byte)	Description
Offset	8	Offset from the top of TUP to the Inner Package.
Size	8	Size of Inner Package.



VH type : TLV format (Version 1)

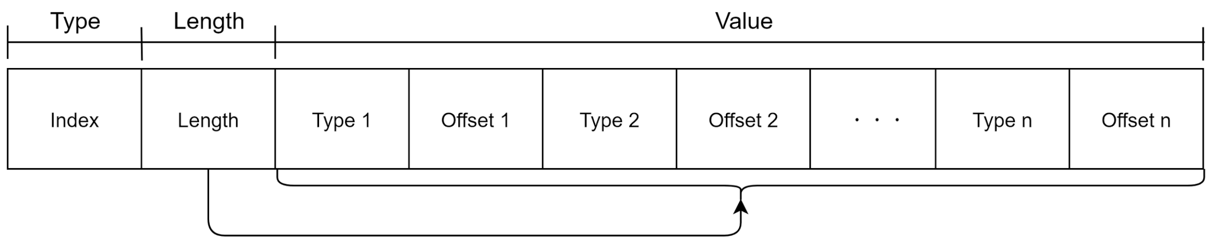


This format contains zero or more data in TLV format within a range that does not exceed the size of VH defined in FH (other TLV format can be contained if necessary).

Type of this TLV format are :

- Index
- Location information of Inner Package

Type details : Index

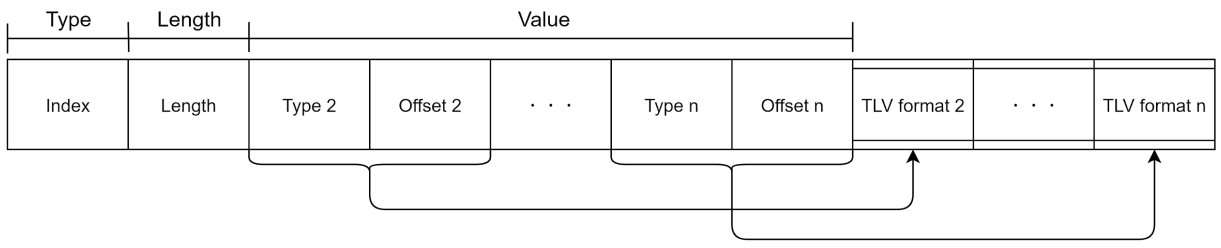


Each TLV format data's type and offset are stored in pairs. Whenever this type exists, it is stored at the beginning of data. Not all the TLV format indexes might be included and the index itself is not included.

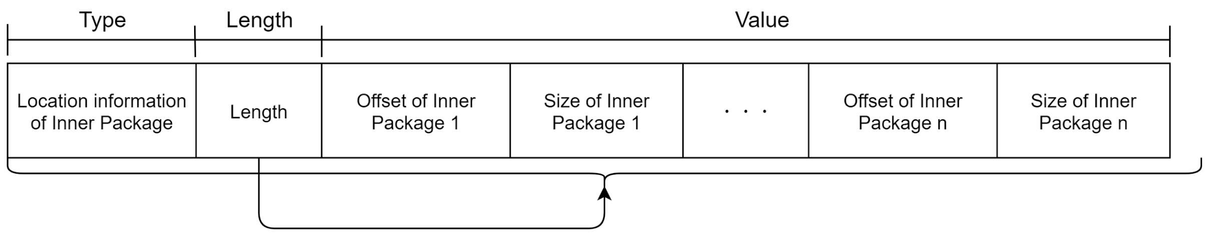
Table. The pair information

Name	Size (byte)	Description
Type	2	Type of the TLV format.
Offset	6	Offset from the top of TUP to the TLV format.

Relationship between Index type and the other TLV formats is like the following.



Type details : Location information of Inner Package


Type
Length
Value
Location information of Inner Package
Length
Offset of Inner Package 1
Size of Inner Package 1
. . .
Offset of Inner Package n
Size of Inner Package n

This format contains every location information of Inner Packages. The offset and the size are contained as the location information of Inner Package in pairs.

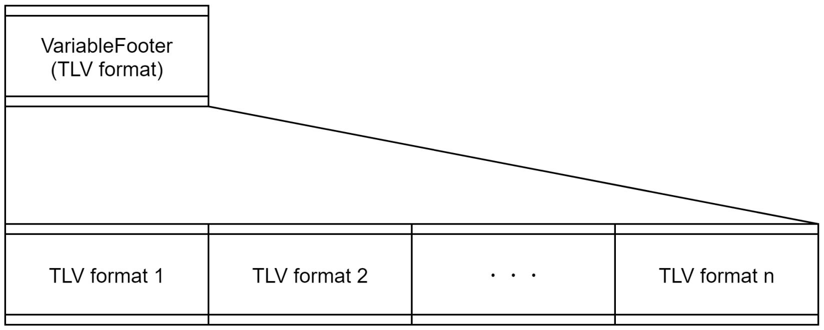
Table. The pair information

Name	Size (byte)	Description
Offset	8	Offset from the top of TUP to the Inner Package.
Size	8	Size of the Inner Package.

VariableFooter (VF)

Data structure of VF is determined by its type defined in the FH. Currently, TLV format only can be used as the VH type.

VH type : TLV format (Version 1)

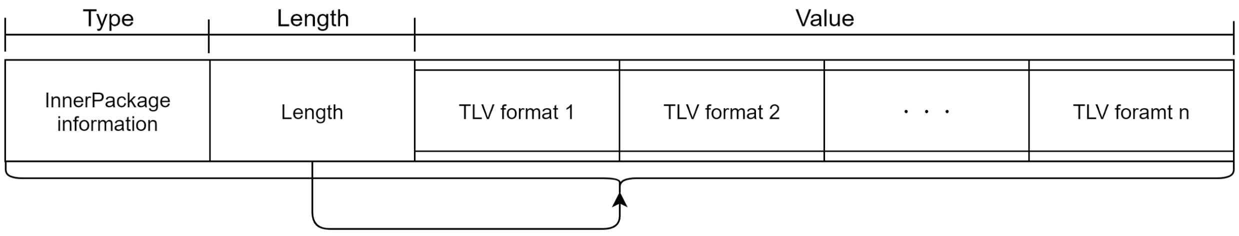


This format contains 0 or more TLV format data within the range of its size defined in the FH.

Types of this TLV format are :

- Index
  - Refer to [Type details : Index]() for the details.
- Inner Package information

Inner Package information

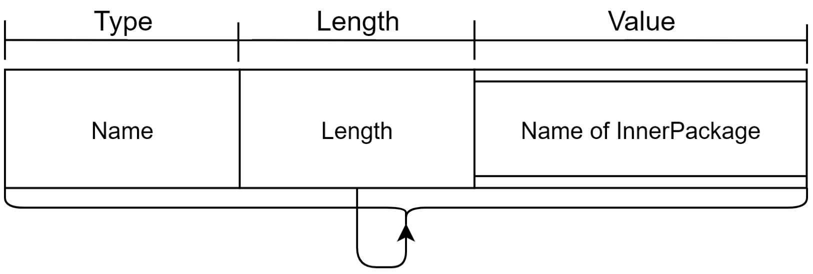


This type contains 0 or more TLV format data with the range of the Length. The order of Inner Packages is the same as [VH type : Fixed Order (Version1)]().

Types of this TLV format are :

- Index
  - Refer to Type details : Index for the details.
- Name
- Version
- Domain
- Compression (whole)
- Compression (fixed size block)
- Compression (variable size block)
- Encryption method
- ICV-TREE
- ICV-RRAY
- Process order (under consideration)
- Update flow (under consideration)
- DELTA-PATCH
- ICV-AFTER-DATA
- ICV-BLOCK

Type details : Name



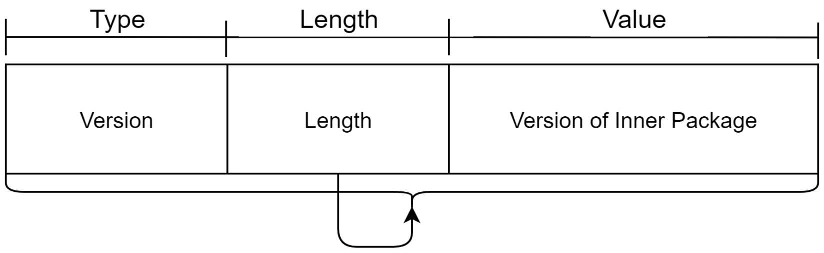
The name of the Inner Package (String) is stored.

Table. Details of the Value

Name	Size (byte)	Description
Name of Inner	variable	The name of Inner Package (not including null

Package		terminator).

Type details : Version

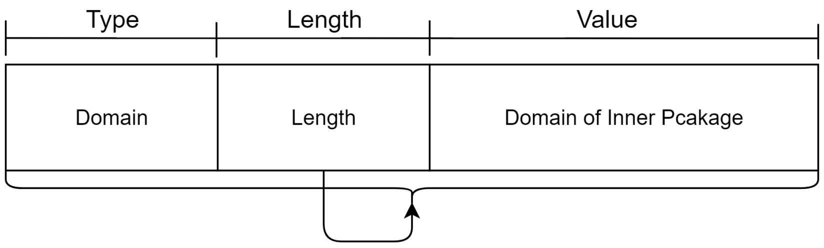


The version of the Inner Package is stored.

Table. Details of the Value

Name	Size (byte)	Description
Version of Inner Package	8	The version of Inner Package.

Type details : Domain



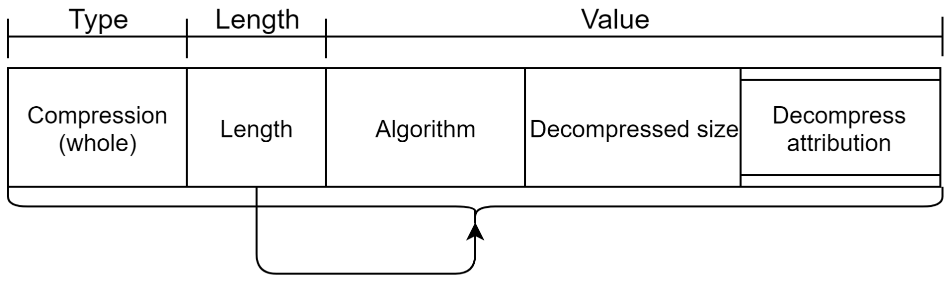
The domain of the Inner Package is stored.

Table. Details of the Value

Name	Size (byte)	Description

Domain of Inner Package	8	The domain that the Inner Package is applied to (such as ECUs).

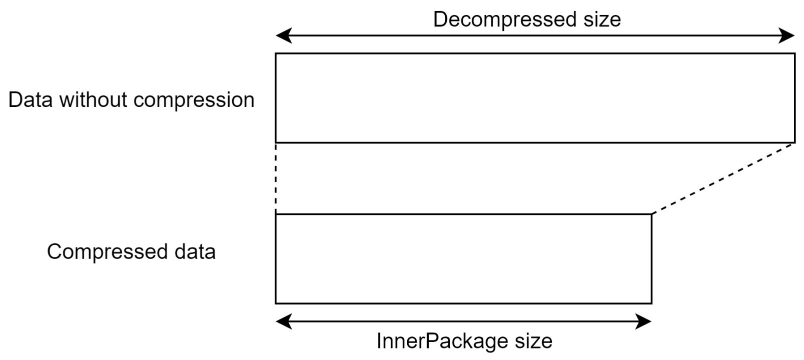
Type details : Compression (whole)



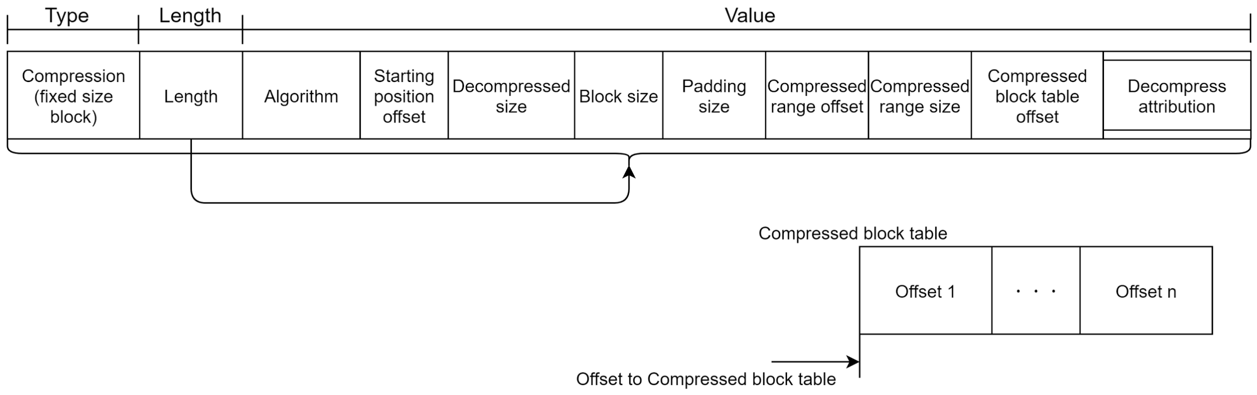
This type exists when the entire Inner Package is compressed at once.

Table. Details of the Value

Name	Size (byte)	Description
Algorithm	8	Compression algorithm.
Decompressed size	8	Size after decompression.
Decompress attribution	variable	Algorithm specific attribution.



Type details : Compression (fixed size block)



This type exists when the target data is compressed with a fixed sized block.

The compressed data can be obtained using compressed range offset and compressed range size. The starting position offset and decompressed size represents the data range after decompression.

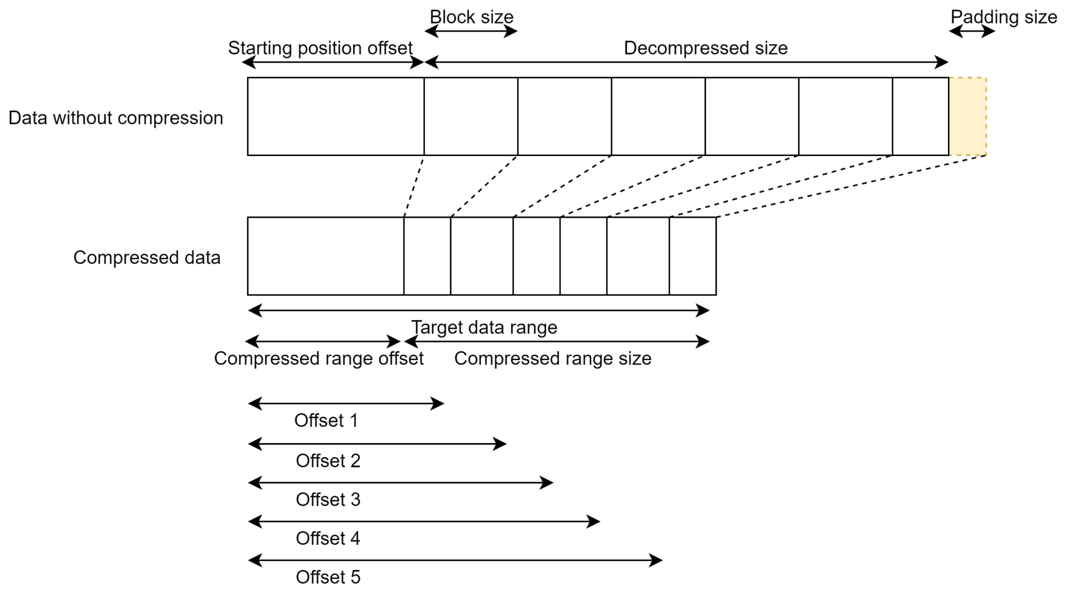
Compressed block tables can be placed at any position in a TUP file and can be accessed using offset to compressed block tables. Random access is possible if there are multiple offsets. If a gap between consecutive offsets is the same as the block size, the block is not compressed.

There can be multiple of this type of information for a target data. In this case, decompressed ranges don’t overlap each other and are stored in a TUP file by ascending order of offsets.

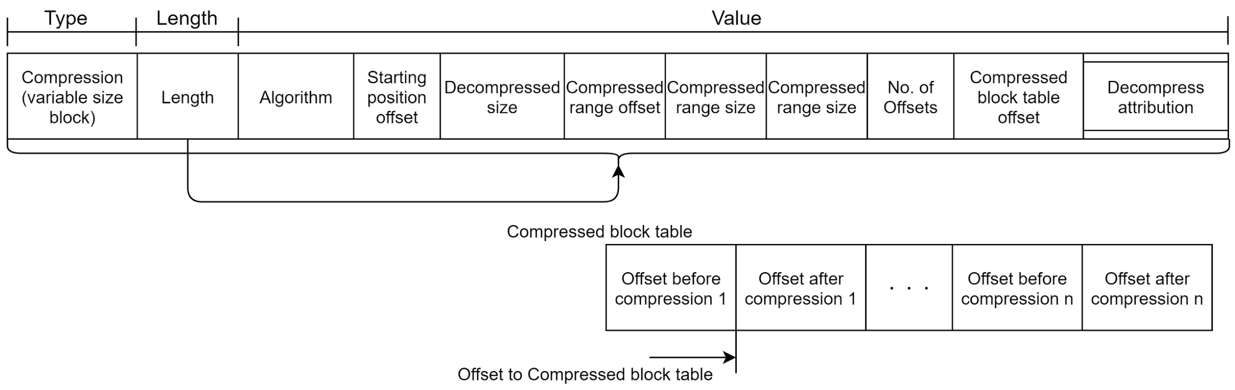
Table. Details of the Value

Name	Size (byte)	Description
Algorithm	8	Compression algorithm.
Starting position offset	8	Offset to the starting position of compression. The format is : ● The top 16 bits are reserved. ● The latter 48 bits are offset value and always positive value.

		• When MSB is 0, the offset is from the top of the TUP file. • When MSB is 1, the offset is from the top of the Inner Package.
Decompressed size	8	Size after decompression.
Block size	4	Unit size of compression. If data is smaller than this size, it is 0-filled by the padding size.
Padding size	4	Padding size of terminal block.
Compressed range offset	8	Offset to compressed range. Refer to format for the details.
Compressed range size	8	Size of compressed range.
Compressed block table offset	8	Offset to a compressed block table. Refer to format for the details.
Decompress attribution	variable	Algorithm specific attribution.
Offset	8	Offset to each compressed block.



Type details : Compression (variable size block)


Type
Length
Value
Compression (variable size block)
Length
Algorithm
Starting position offset
Decompressed size
Compressed range offset
Compressed range size
Compressed range size
No. of Offsets
Compressed block table offset
Decompress attribution
Compressed block table
Offset before compression 1
Offset after compression 1
. . .
Offset before compression n
Offset after compression n
Offset to Compressed block table

This type exists when the target data is compressed with a variable sized block.

The compressed data can be obtained using compressed range offset and compressed range size. The starting position offset and decompressed size represents the data range after decompression.

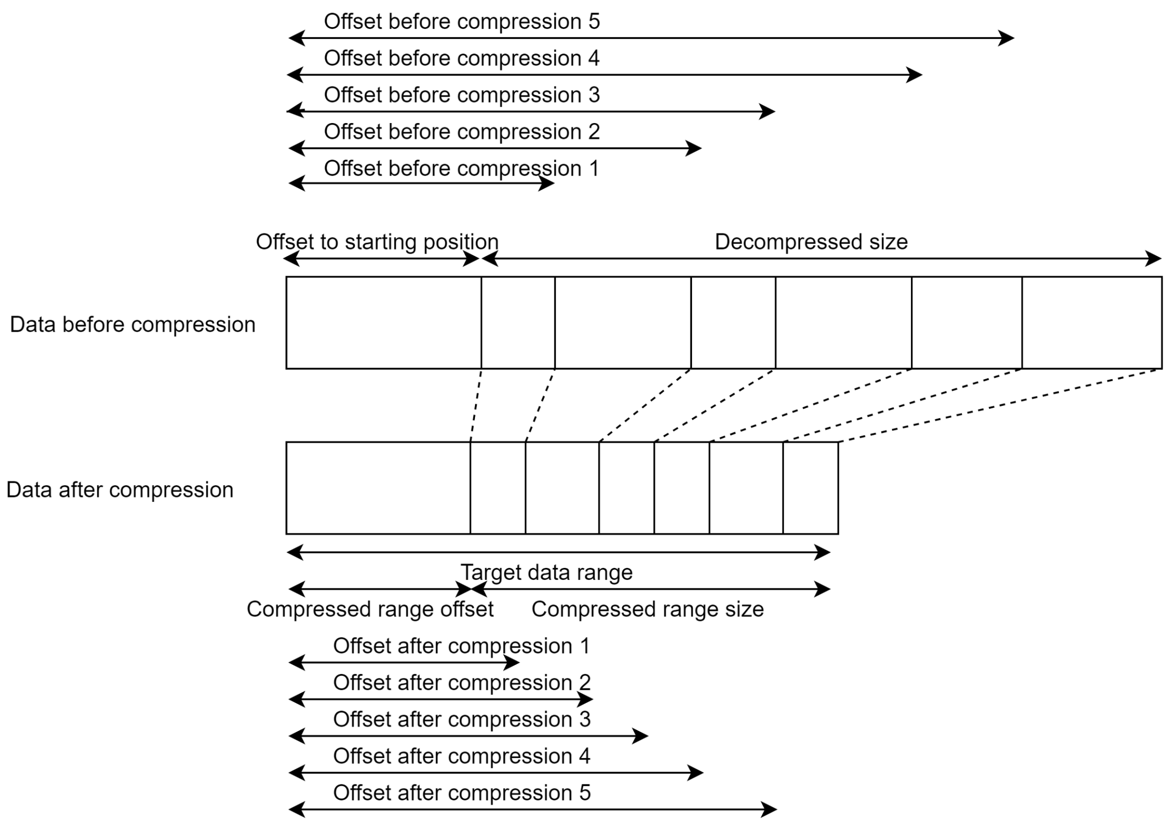
Compressed block tables can be placed at any position in a TUP file and can be accessed using offset to compressed block tables. The number of offsets is the number of offsets in the table. Random access is possible if there are multiple offsets. If a gap between consecutive offsets is the same as the block size, the block is not compressed.

There can be multiple of this type of information for a target data. In this case, decompressed ranges don't overlap each other and are stored in a TUP file by ascending order of offsets.

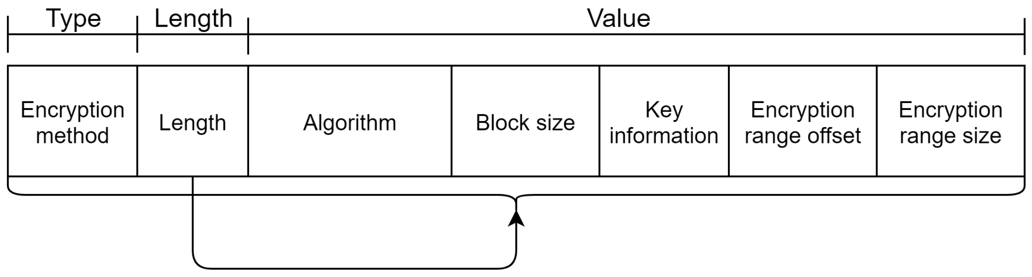
Table. Details of the Value

Name	Size (byte)	Description
Algorithm	8	Compression algorithm.
Starting position	8	Offset to the starting position of compression. Refer to format for the details.
Decompressed size	8	Size after decompression.
Compressed range	8	Offset to compressed range. Refer to format for

offset		the details.
Compressed range size	8	Size of compressed range.
The number of offsets	8	The number of offsets in the compressed block table.
Compressed block table offset	8	Offset to a compressed block table. Refer to format for the details.
Decompress attribution	variable	Algorithm specific attribution.
Offset before compression	8	Offset to each block before compression. Refer to format for the details.
Offset after compression	8	Offset to each block after compression. Refer to format for the details.


Offset before compression 5
Offset before compression 4
Offset before compression 3
Offset before compression 2
Offset before compression 1
Offset to starting position
Decompressed size
Data before compression
Data after compression
Target data range
Compressed range offset
Compressed range size
Offset after compression 1
Offset after compression 2
Offset after compression 3
Offset after compression 4
Offset after compression 5

Type details : Encryption method

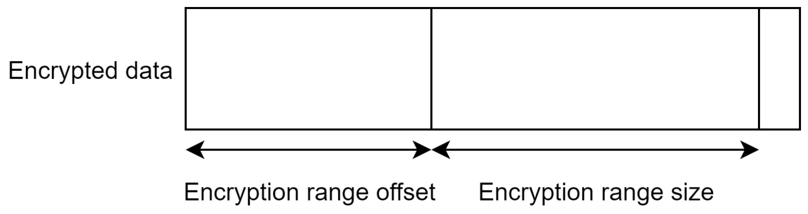


This type exists when the target data is encrypted.

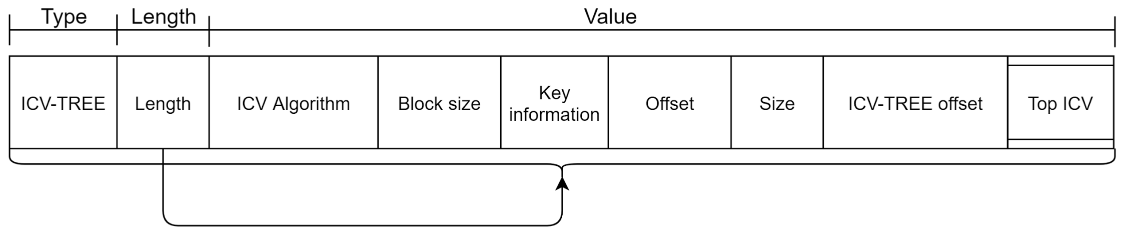
Encrypted data can be found using encryption range offset and encryption range size. It is encrypted using a key which can be obtained from the key information.

Table. Details of the Value

Name	Size (byte)	Description
Algorithm	8	Encryption algorithm.
Block size	8	Block size of encryption.
Key information	8	Information to obtain a key to decrypt.
Encryption range offset	8	Offset to the starting position of encryption. Refer to format for the details.
Encryption range size	8	Size of encrypted range.



Type details : ICV-TREE



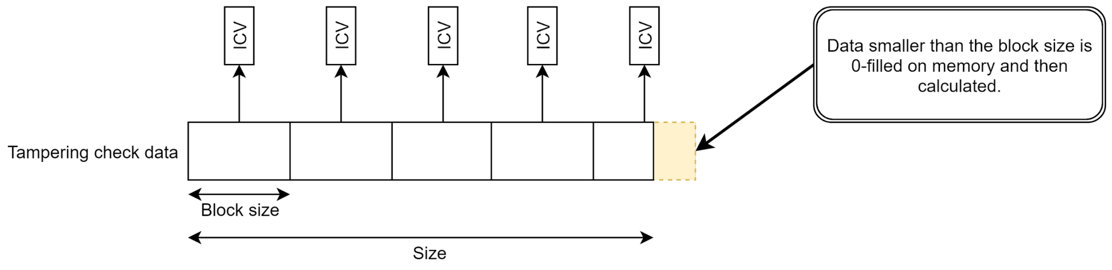
This type is for information on ICV tree structure to check tampering.

Table. Details of the Value

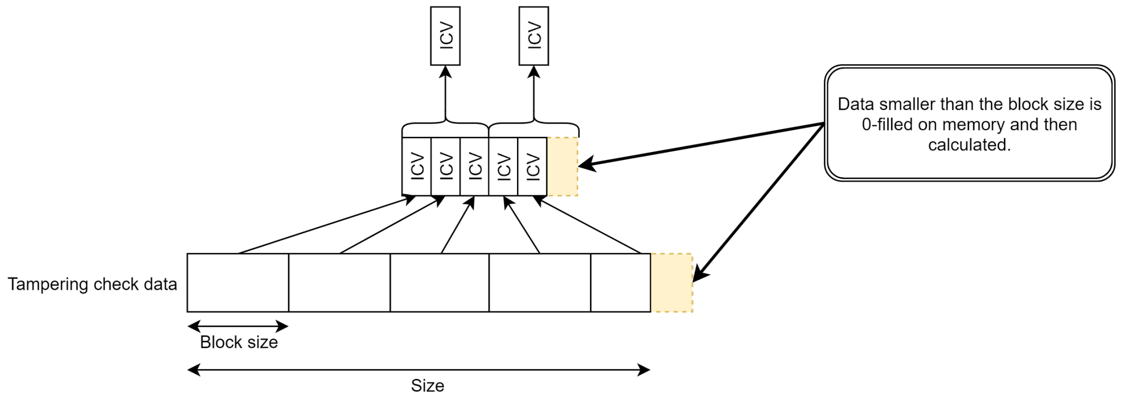
Name	Size (byte)	Description
ICV algorithm	4	ICV generation algorithm.
Block size	4	Unit size to generate one ICV. If data is smaller than this size, it is 0-filled by the padding size.
Key information	8	Information to obtain a key to generate ICV.
Offset	8	Offset from the top of a TUP file to the starting position of tampering check data.
Size	8	Size of tampering check data.
ICV-TREE offset	8	Offset from the top of a TUP file to the ICV-TREE data.
Top ICV	Depends on ICV algorithm	Top ICV of the ICV-TREE

ICV generation flow

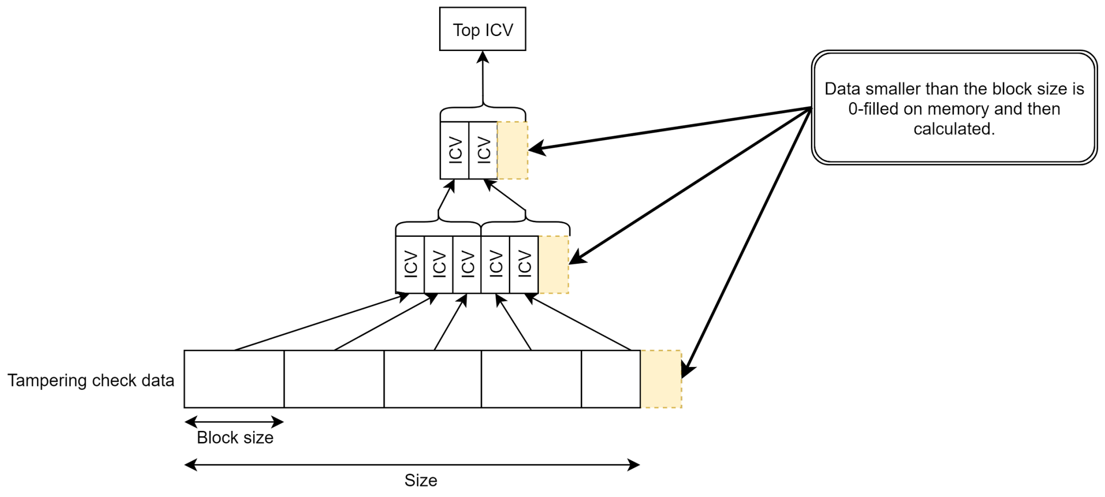
1. Obtain tampering check data using the offset and the size, and then divide it by the block size. Generate ICV of each divided data with a key obtained from the ICV algorithm and key information. Apply 0-fill padding to data that is smaller than the block size. Gather generated ICVs and connect them to ICV-TREE data.


ICV
ICV
ICV
ICV
ICV
Data smaller than the block size is 0-filled on memory and then calculated.
Tampering check data
Block size
Size

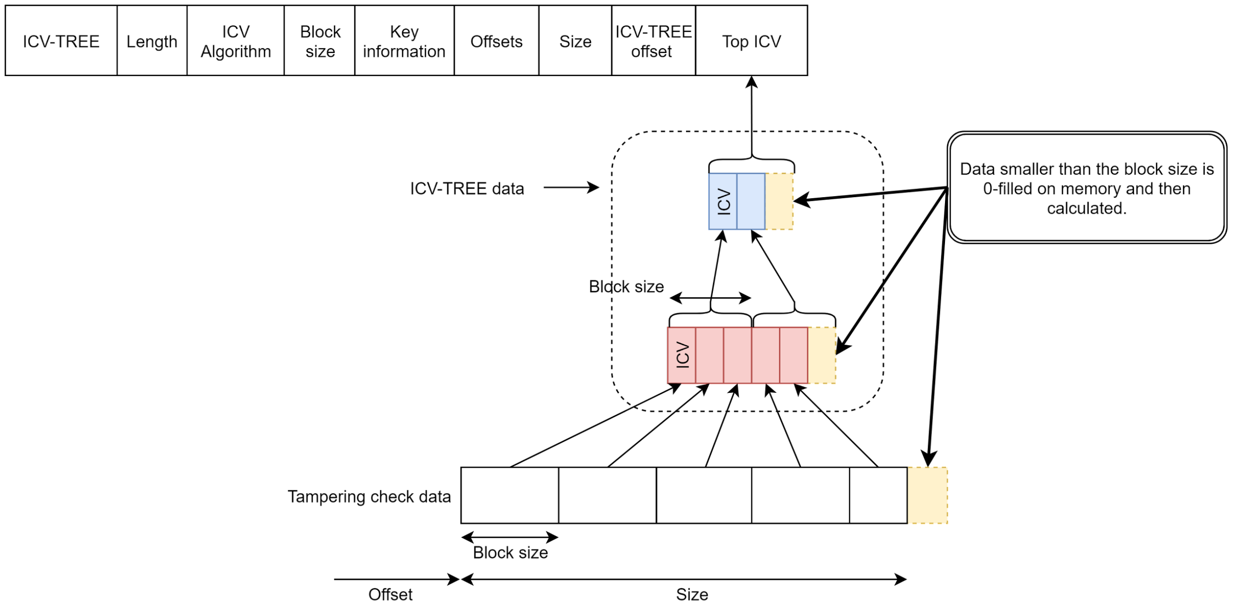
2. Divide gathered ICVs by the block size and generate ICV with a key obtained from the ICV algorithm and key information.

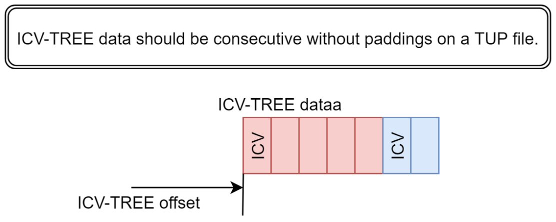

ICV
ICV
ICV
ICV
ICV
ICV
ICV
Data smaller than the block size is 0-filled on memory and then calculated.
Tampering check data
Block size
Size

3. Repeat this process until only one ICV will be generated. The last one ICV is the Top ICV.

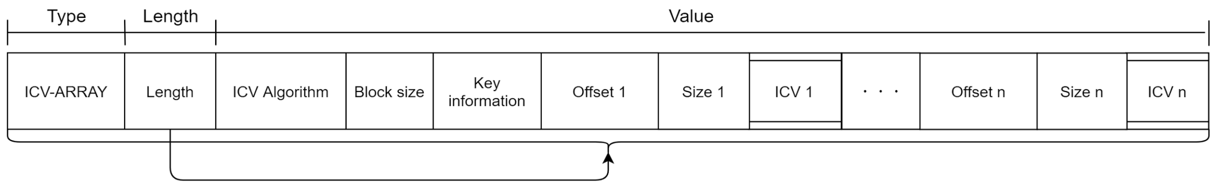

Top ICV
ICV
Data smaller than the block size is 0-filled on memory and then calculated.
Tampering check data
Block size
Size

ICV-TREE data excluding Top ICV is placed at any position of a TUP file. ICV-TREE offset is for the place of the ICV-TREE data.




ICV-TREE data should be consecutive without paddings on a TUP file.
ICV-TREE dataa
ICV
ICV-TREE offset

Type details : ICV-ARRAY



This type summarizes multiple tampering check data.

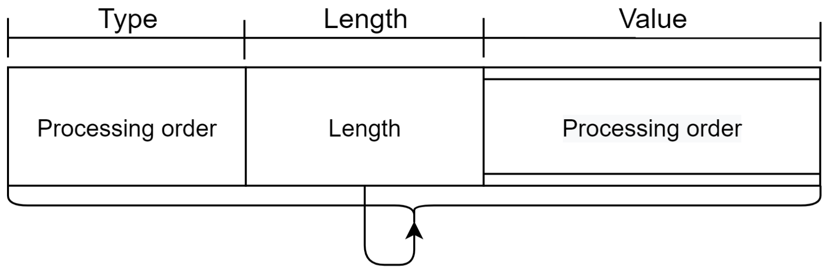
The tampering check data includes ICV-ARRAY and ICV-TREE too. The ICV can be calculated by the tampering check data using a key obtained from the ICV algorithm and key information.

Table. Details of the Value

Name	Size (byte)	Description
ICV algorithm	4	ICV generation algorithm.
Block size	4	Unit size to generate one ICV. If data is smaller than this size, it is 0-filled by the padding size.
Key information	8	Information to obtain a key to generate ICV.
Offset	8	Offset from the top of a TUP file to the starting position of tampering check data. The format is : ● The top 16 bits are reserved. ● The latter 48 bits are offset value and always positive value. ● When MSB is 0, it indicates raw data. ● When MSB is 1, it indicates ICV-TREE or ICV-ARRAY in TLV format.
Size	8	Size of tampering check data.

ICV	Depends on ICV algorithm	ICV for tampering check data.

Type details : Processing order (under consideration)

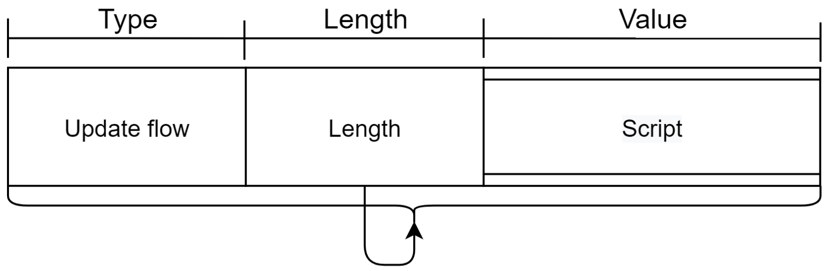


This type contains a processing order when a default order, that is compress, encrypt and generate ICV, is not applied.

Table. Details of the Value

Name	Size (byte)	Description
Processing order	4	Processing order other than default order.

Type details : Update flow (under consideration)

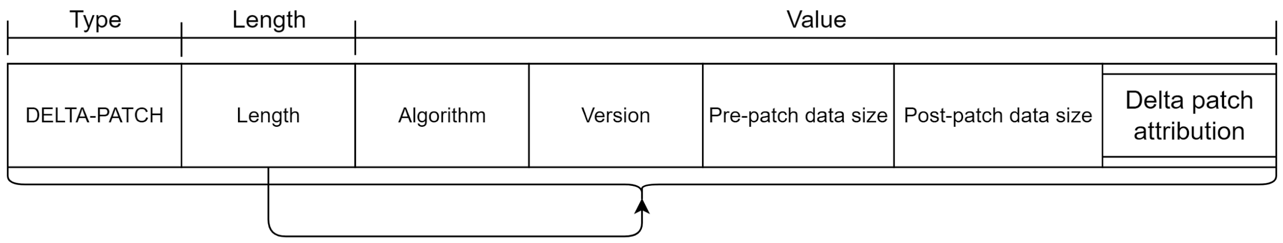


This type contains a script and an update flow of Inner Packages are written in it.

Table. Details of the Value

Name	Size (byte)	Description
Script	variable	Update flow of Inner Package.

Type details：DELTA-PATCH

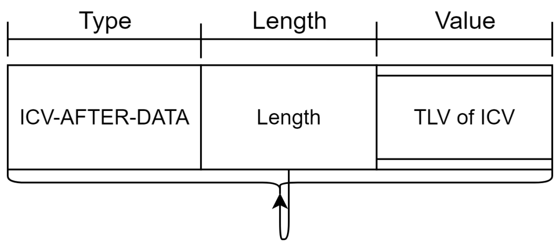


This type exists when the target data is a delta patch.

Table. Details of the Value

Name	Size (byte)	Description
Algorithm	4	Delta patch algorithm.
Version	4	Delta patch algorithm version.
Pre-patch data size	8	Size of pre-patch data.
Post-patch data size	8	Size of patched data.
Delta patch attribution	variable	Algorithm specific attribution.

Type details：ICV-AFTER-DATA

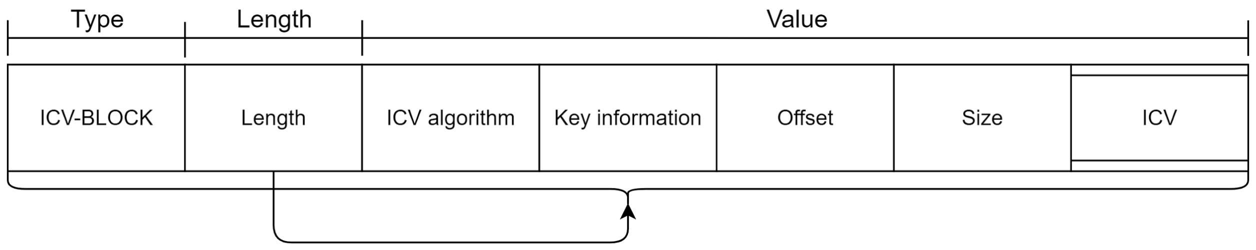


This type is ICV information for patched data.

Table. Details of the Value

Name	Size (byte)	Description
TLV of ICV	variable	ICV-TREE or ICV-BLOCK.

Type details：ICV-BLOCK



This type is ICV block information to check for tampering.

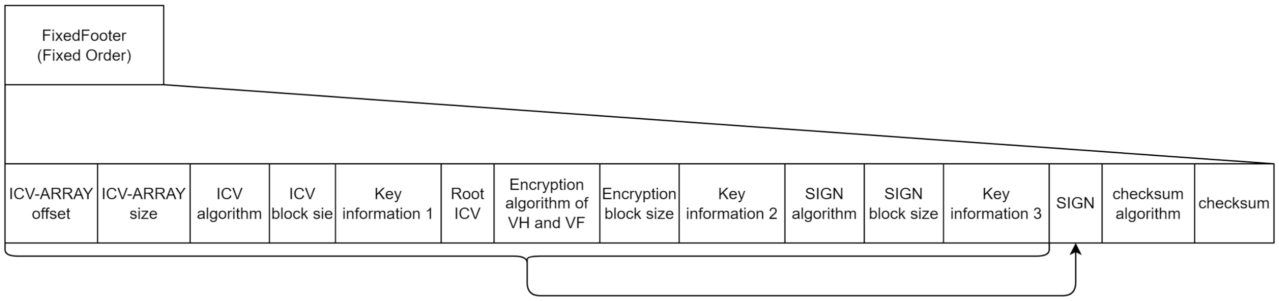
Table. Details of the Value

Name	Size (byte)	Description
ICV algorithm	8	ICV generation algorithm.
Key information	8	Information to obtain a key to generate ICV.
Offset	8	Offset from the top of a TUP file to the starting position of tampering check data.
Size	8	Size of tampering check data.
ICV	Depends on ICV algorithm	ICV for tampering check data.

FixedFooter (FF)

Data structure of FF is determined by the FF type in the FH. Currently, only the ‘Fixed Order’ can be used for ‘FF type’.

FF type : Fixed Order (Version 1)



This type has the Root ICV of each ICVs and the Root ICV is calculated by ICV-ARRAY with a key obtained from the ICV algorithm and key information 1. ICV-ARRAY offset, ICV-ARRAY size and ICV algorithm are included in the tampering check target of ICV-ARRAY.

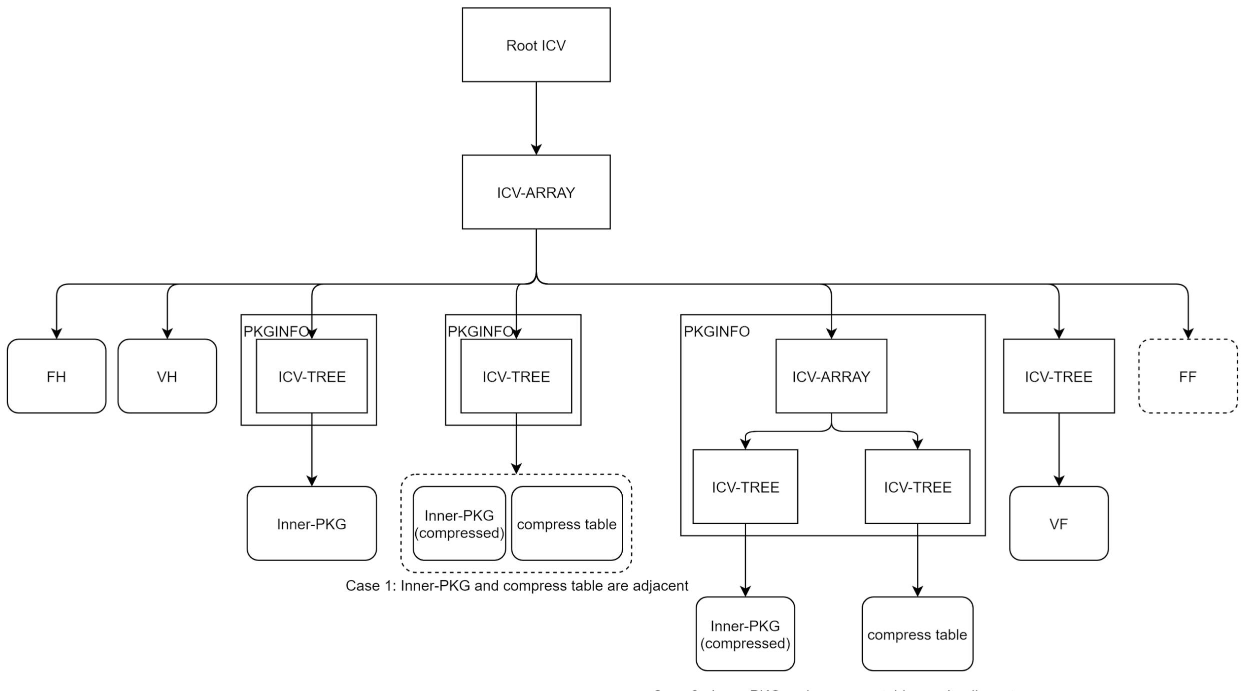
Signature of the Root ICV is calculated by the data of between the top of the FF and the SIGN with a key obtained from the SIGN algorithm and key information 3.

Checksum to brief-check tampering data in the entire TUP file is calculated by the checksum algorithm.

Table. Data structure of FH

Name	Size (byte)	Description
ICV-ARRAY offset	8	Offset from the top of a TUP file to ICV-ARRAY.
ICV-ARRAY size	8	Size of ICV-ARRAY indicated by the ICV-ARRAY offset.
ICV algorithm	4	Algorithm to generate the Root ICV.
ICV block size	4	Unit size to generate one ICV. If data is smaller than this size, it is 0-filled by the padding size.
Key information 1	64	Information to obtain a key to generate the Root ICV. The actual size depends on the ICV algorithm.

Root ICV	128	Root ICV of each ICVs. Calculated by the ICV-ARRAY. The actual size depends on the ICV algorithm and unused bytes are 0-filled.
Encryption algorithm of VH and VF	4	Encryption algorithm used for encryption of VH and VF.
Encryption block size	4	Block size of encryption.
Key information 2	64	Information to obtain a key used for decryption. The actual size depends on the encryption algorithm of VH and VF. (The unused bytes are 0-filled)
SIGN algorithm	4	Algorithm to generate SIGN.
SIGN block size	4	Block size to generate SIGN.
Key information 3	64	Information to obtain a key to generate SIGN. The actual size depends on the SIGN algorithm. (The unused bytes are 0-filled)
SIGN	Depends on SIGN algorithm	Calculated by the data between the top of FF and SIGN.
checksum algorithm	4	Algorithm to generate checksum.
checksum	4	Data to check tampering of the entire TUP file.



Case 1: Inner-PKG and compress table are adjacent

Case 2 : Inner-PKG and compress table aren't adjacent