

Huawei DOPRA SSP
V300R005C00SPC123B200

Security Target

Issue 1.3
Date 2023-03-29



Copyright © Huawei Technologies Co., Ltd. 2022. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People’s Republic of China

Website: <http://www.huawei.com>

Email: support@huawei.com

About This Document

Purpose

This document provides description about ST (Security Target).

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Date	Revision Version	Section Number	Change Description	Author
2019-07-30	0.1	ALL	Initial Draft	iangjicheng liubin
2019-09-17	0.2	ALL	CC WorkShop Review Merge	iangjicheng iangj wugang
2019-10-25	0.3	1.3.1 1.3.2 1.4.3 3.1~3.3 3.4.3 4.3.1~4.3.2 5.4.1~5.4.2	Reply to Ors of ASE 2019.09.27	iangjicheng Liubin
2019-11-13	0.4	1.3.2	Reply to Ors of ADV 2019.10.26	wugang
2019-12-10	0.5	ALL	Reply to comments-1202	Liangjicheng
2020-2-14	0.51	ALL	Reply to comments-0207	Liangjicheng
2020-2-26	0.52	1.3.1.1 1.3.2 1.3.3 1.4.3	Reply to comments-0207 according to	Liangjicheng

Date	Revision Version	Section Number	Change Description	Author
		3.2 3.3 4.3.1 5.2.1.2 6.4	discussion	
2020-3-18	0.6	1.3 1.4	Reply to Comments-0316	Liangjicheng
2020-4-24	0.61	1.3.1.1 1.3.2 3.1 3.2 3.3	Reply to Comments-0402	Liangjicheng
2020-7-2	0.70	4.3 5.4	Reply to OR-03 OR-04	Liangjicheng
2020-7-22	0.71	3.1~3.3 4.1 4.3.2 5.4.2		Liangjicheng
2020-7-24	0.72	All	Delete comment	Liangjicheng
2020-8-13	0.8	3.1 3.3 4.1 4.3.1 4.3.2 5.4.2		Liangjicheng Wugang
2020-8-25	0.81	3.1 3.2 3.2 4.1 4.3.1 4.3.2 5.4.1 5.4.2 6.2 6.3 6.4	Update Assert Threat	Liangjicheng Wugang
2020-9-8	0.82	1.3.1.2 1.4.2 4.1 4.3.1 4.3.2 5.4.1 5.4.2 6.2~6.5		Liangjicheng Wugang
2020-9-30	0.9	All section	Replay to V1.1 OR OR-01 02 03 04 05 19	Liangjicheng Wugang
2020-10-11	0.91	4.1 6.2 6.3	Replay to V1.1 OR OR-04 05	Wugang
2020-12-19	1.0	1.3.1.1 1.3.2 1.4.1 1.4.2 3.1 3.3 4 5 6		Liangjicheng wugang
2021-03-15	1.0-0315	6.1	Updating the Black Box Parsing Tool	wugang
2021-03-18	1.0-0317 1.0-0318	1.4.1	Add parsing tool package Update checksum	liangjicheng

Date	Revision Version	Section Number	Change Description	Author
2021-07-15	1.0-0715	1.3.3 3.4.3 4.2 6.1 6.4	Modify base on review commends	liangjicheng
2021-07-16	1.0-0716		Modify base on review commends	liangjicheng
2021-07-21	1.0-0721	1.3.1 1.4.1 4.1 5.2 5.3 6.1	Delete FAU_SAR.1 FAU_SAR.3	liangjicheng
2021-09-21	1.0-0921	4.3 5.4 6.4	Update according to OR	liangjicheng
2021-10-07	1.0-1007	5	Update according to OR	liangjicheng
2021-10-07	1.0-1109	3.1 6.4 5.2.4.9	Update according to OR	wugang
2021-11-12 2021-12-14	1.00	6.2.3 6.2.4 6.3	Add new SFRs according to OR, and update related tables	wugang
2021-12-28	1.0-1228	3.1 A Acronyms and Abbreviations C Appendix: Technical of CRC	Update according to 20211221_DOPR A_ASE_AGD_Comments	wugang
2022-01-05	1.0-0105	3.1 A Acronyms and Abbreviations C Appendix: Technical of CRC	Update according to 20211221_DOPR A_ASE_AGD_Comments	wugang
2022-01-06	1.0-0106	1.4.1 1.4.3	Update the Software and Documents for SDK OPE PRE.	Liangjicheng

Date	Revision Version	Section Number	Change Description	Author
2022-01-20	1.0-0120	1.3.1.1 3.1.1 3.1.2 3.1.3 3.1.4 3.3.4 4.1 Table 5 4.2 Table 6 O.BB.PROTEC TION 4.3.1 Table 8 P.EVENTS 6.2.1.1 7.4	A.RESOURCE OE.RESOURCE Delete file parse Delete event: Memory dumping/ Black Box file damaged	liangjicheng
2022-03-03	1.0-0303	3.1.3 Black box and black box file (The structure of an audit record is)	[20220221]20220 104_HUA_DOPRA _ASE_AGD_DEKR A_Comments	wugang
2022-03-19	1.0-0319	1.3.3 1.4.3	About borad type	liangjicheng
2022-04-13	1.0-4.13	Appendix C	CRC	
2022-04-13	1.0-4.13	1.4.1 1.3.1.2 1.4.2 3.2.4 4.3.1 6.2.2.1 6.2.4 7.5	1. CPU overload Task dead loop Delete 2. update 1.4.1	liangjicheng
2022-04-19	1.0-4.19	Appendix C	CRC calculation process improved	huawei
2022-04-19	1.0-4.22	1.4.2 4.1 4.2 4.3.1 Table 8 6.2.1 7.2 7.3 1.3.3 1.4.3	1 Remove event “Read error value from configuration file” 2 VM RAM	liangjicheng
2022-04-25	1.0-4.25	3.1.2	Reply 20220422_ATE_D	wugang

Date	Revision Version	Section Number	Change Description	Author
			EKRA_Comments	
2022-05-12	1.0-5.12	C.1	Description of memory CRC calculation range	wugang
2022-06-01	1.0-6.1		Change FAU_SAA.3 to FAU_SAA.1 Delete Black Box memory damaged	Liangjicheng
2022-06-06	1.0-0606	6.2.3, 7.4	Remove black box memory damaged from FDP_SDI.2	Huawei
2022-06-10	1.0-0610	1.3.1 1.4.2 4.1 7.4	Remove memory leakage detection from security objective, and logical scope. Remove black box memory damage from logical scope and TSS. Remove CPU resources monitoring from security objective. Correct description in FDP_SDI.2	Huawei
2022-06-10	1.1	1.1	Final version	Huawei
2023-02-20	1.2	1.1, 1.4.1	Final version	Huawei
2023-03-29	1.3	1.1, 1.4.1	Final version	Huawei

Contents

About This Document	ii
1 Introduction	1
1.1 ST reference	1
1.2 TOE reference	1
1.3 TOE overview	1
1.3.1 TOE usage and major security features	1
1.3.1.1 TOE Usage	1
1.3.1.2 TOE major security features	2
1.3.2 TOE type	2
1.3.3 Non-TOE Hardware and Software	3
1.4 TOE description	5
1.4.1 Physical scope of the TOE	5
1.4.2 Logical scope of the TOE	6
1.4.3 Evaluated configuration	7
2 Conformance claims	8
2.1 CC Conformance Claim	8
3 Security Problem Definition	9
3.1 Assets	9
3.1.1 Memory partition	9
3.1.2 Message partition and UIPC Message	11
3.1.3 Black box and black box file	13
3.2 Organizational Security Policies (OSP)	16
3.2.1 P.EVENTS	17
3.2.2 P.INTEGRITY	17
3.2.3 P.ALARM	17
3.2.4 P.RESOURCE.UTILIZATION	17
3.3 Assumptions	17
3.3.1 A.PHYSICAL	17
3.3.2 A.NOEVIL	17
3.3.3 A.INSTALL	17
3.3.4 A.RESOURCE	17

3.3.5 A.OS	17
3.3.6 A.TIME	18
4 Security Objectives	18
4.1 Security Objectives for the TOE	18
4.2 Security Objectives for the Operational Environment	19
4.3 Security Objectives rationale	20
4.3.1 Coverage	20
5 Extended Components Definition	23
6 Security Requirements for the TOE	24
6.1 Conventions	24
6.2 Security Functional Requirements	24
6.2.1 Security Audit (FAU)	25
6.2.1.1 FAU_GEN.1 Audit data generation	25
6.2.1.2 FAU_SAA.1 Potential violation analysis	25
6.2.1.3 FAU_ARP.1 Security alarms	25
6.2.1.4 FAU_STG.4 Prevention of audit data loss	25
6.2.2 Resource Utilization (FRU)	25
6.2.2.1 FRU_PRS.1 Limited priority of service	25
6.2.2.2 FRU_RSA.2 Minimum and maximum quota	26
6.2.3 User Data Protection (FDP)	26
6.2.3.1 FDP_SDI.2 Stored data integrity monitoring and action	26
6.2.3.2 FDP_UIT.1 Data exchange integrity	26
6.2.3.3 FDP_IFC.1 Subset information flow control	26
6.2.3.4 FDP_IFT.1 Simple security attributes	26
6.3 Security Requirements Dependency Rationale	27
6.4 Security Functional Requirements Rationale	28
6.4.1 Coverage	28
6.4.2 Sufficiency	29
6.5 Security Assurance Requirements	30
6.6 Security Assurance Requirements Rationale	31
7 TOE Summary Specification	32
7.1 Auditing	32
7.2 Message Protection	32
7.3 Memory monitoring	32
7.4 Black Box Protection	33
7.5 Resource Utilization	33
A Acronyms and Abbreviations	35
B Technical References	37
C Appendix: Technical of CRC	38

Figures

Figure 1 TOE and its environment	3
Figure 2 Hardware and Software	3

Tables

Table 1 IT Environment Components	4
Table 2 Supported Configuration for OS + CPU	4
Table 3 Physical Scope	6
Table 4 Evaluated configuration	7
Table 5 Security Objectives for the TOE	18
Table 6 Security Objectives for the Operational Environment	19
Table 7 Security Objectives coverage	20
Table 8 Security Objectives rationale	21
Table 9 Security Functional Requirements	24
Table 10 Dependencies between TOE Security Functional Requirements	27
Table 11 Security Objectives Coverage	28
Table 12 SFR sufficiency analysis	29
Table 13 Security Assurance Requirements	30

1 Introduction

This section contains the ST reference, TOE reference, TOE overview and TOE description of **Huawei DOPRA SSP**.

1.1 ST reference

This ST is uniquely identified as below,

Title: Huawei DOPRA SSP V300R005C00SPC123B200 Security Target

Version: 1.3

Author: Huawei Technologies Co., Ltd.

Publication date: 2023-03-29

1.2 TOE reference

The TOE is identified as below,

TOE name: Huawei DOPRA SSP

TOE version: V300R005C00SPC123B200

Programmer: Huawei Technologies Co., Ltd.

1.3 TOE overview

This section summarizes the TOE usage and the TOE major security features, the TOE type and the Non-TOE hardware/software required by the TOE to operate.

The TOE is a library called DOPRA SSP. The ST contains a description of the security objectives and the requirements, as well as the necessary functional and assurance measures provided by the TOE.

The ST provides the basis for the evaluation of the TOE according to the Common Criteria for Information Technology Security Evaluations (CC).

In this document, the acronym APP refers to the Customer Application Service which is linked to the TOE and use the TOE.

1.3.1 TOE usage and major security features

1.3.1.1 TOE Usage

The TOE is widely used in Huawei products. It is a library which can be dynamically linked with a designated APP. The TOE has the following capabilities:

- Provides secondary memory management for APP, in which the memory monitoring functionality is implemented.
- Provides message management mechanism for communication between APP processes, in which the message protection functionality is implemented.
- Provides black box mechanism to ensure that audit log is not lost when a process is reset, including segment-based management, file dumping.
- Provides log management mechanism for system security event logging and reading, in which the auditing functionality is implemented.
- Provides basic system monitoring capability in which resource protection functionality is implemented.

1.3.1.2 TOE major security features

The major security features implemented by the TOE and subject to evaluation are:

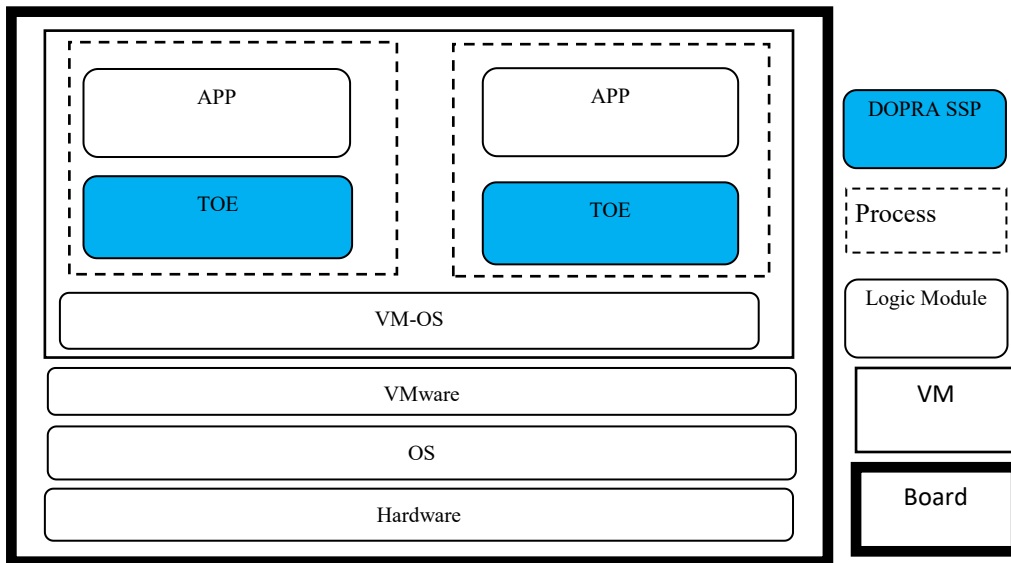
- **Auditing**
The TOE records audit logs for detected security events.
- **Message Protection**
The TOE monitors and verifies the incoming and outgoing messages. Generates a security log with an associated alarm severity level and sends an alarm to the user.
- **Memory monitoring**
The TOE monitors the memory. Generates a security log with an associated alarm severity level and sends an alarm to the user.
- **Black Box Protection**
The black box monitors the internal memory and the file system of the TOE, transfers logs from memory to files (dumping mechanism).
- **Resource utilization**
The TOE detects system resources (memory) overload and message flow control mechanism. Allocates resources for each APP process.

1.3.2 TOE type

The TOE is library, which provide an SDK with C Head Files(*.h) and C Library files (*.so) and can run on various hardware platforms (such as X86 ARM PPC) and operating systems (such as: VxWorks Linux).

The TOE and its environment are shown in Figure 1.

Figure 1 TOE and its environment



In Figure 1, the blue box indicates the position of the TOE. The TOE and APP work together in one process, and all the processes are running on the same OS in one VM board. The APP, VM-OS, VMware, OS and Hardware belong to the TOE environment, and are not the part of TOE.

1.3.3 Non-TOE Hardware and Software

The TOE is a library, it can run on multiple hardware platforms and operating systems. The APP, OS, Driver and Hardware belong to the TOE environment.

Figure 2 Hardware and Software

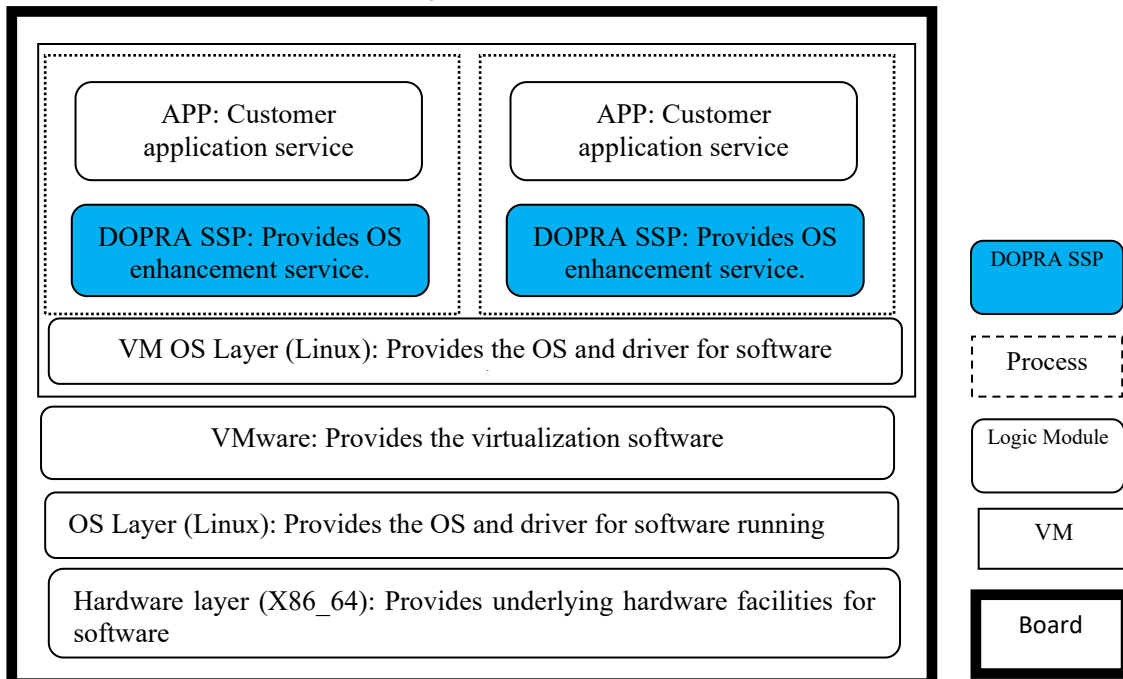


Table 1 IT Environment Components

Component	Required/Optional	Usage/Purpose Description for TOE performance
CPU	Required	CPU supported by the TOE are: X86, ARM, PPC, MIPS. The CPU used for the evaluation is X86_64 (x86 64-bit).
Board	Required	The used board is PC (X86-64bit)
OS layer	Required	Debian GNU/Linux 11 (bullseye)
VM	Required	VMware station 16
VM Hard Disk	Required	Minimum 2GB
VM RAM	Required	Minimum 2GB
VM OS layer	Required	The OS layer provides basic drivers and kernel services. It supports Linux and Vxworks. The OS version used for the evaluation is suselinux12 (SUSE Linux Enterprise Server 12 SP4).
APP	Required	Customer Application Service (developed by the customer) which is linked to the TOE and use the TOE

The DOPRA SSP can run on multiple operating systems and CPU architecture platforms. TOE's evaluation is made using a combination of [suselinux12 + X86_64].

Table 2 Supported Configuration for OS + CPU

Type	Configuration
OS + CPU	Vxworks5.5 + PPC8260 Vxworks5.5 + PPC8321 Vxworks5.5 + ARM926 Vxworks5.5 + ARM-IXP2350 Vxworks5.5 + XLS416 Vxworks6.4 + ARM926 Vxworks6.4 + PPC1010 Vxworks6.4 + PPC8321 Vxworks6.4 + PPC8360 Vxworks6.4 + PPC860 Vxworks6.8 + HI1210(HaiSi) Vxworks6.8 + ARM-ARCH7 Vxworks6.8 + Cavium65xx Vxworks6.8 + XLP316 euler release 2.0 (SP8) (Huawei) + ARM64 euler release 2.0 (SP8) (Huawei) + HI1620(HaiSi)

Type	Configuration
	rtos207.3.2.B012(Huawei) + ARM32 rtos207.3.3.B001(Huawei) + ARM64 rtos207.5.RC100.B001(Huawei) + PPC32 rtos207.3.3.B001 (Huawei) + X86_64 rtos207.3.6.B003 (Huawei) + HI1210(HaiSi) rtos207.3.2.B012 (Huawei) + HI1212(HaiSi) rtos207.3.2.B012 (Huawei) + HI1215(HaiSi) rtos207.3.6.B003 (Huawei) + HI1382(HaiSi) rtos207.3.6.B003 (Huawei) + HI1383(HaiSi) suselinux12 + X86 suselinux12 + X86_64 wrlinux30 + CN5540 wrlinux30 + PPC7447 wrlinux30 + PPC8320 wrlinux30 + PPC8541 wrlinux30 + PPC8572 wrlinux43 + PPC4080 wrlinux43 + PPC5040 wrlinux43 + XLP832 wrlinux60 + ARM5855 wrlinux60 + PPC1010 wrlinux60 + PPC3041 wrlinux42 + HI1380(HaiSi) wrlinux43 + ARM-ARCH7 wrlinux43 + HI1381(HaiSi)

 **NOTE**

The TOE environment components are not part of the TOE, and there is no assurance regarding these components, they will not be evaluated.

1.4 TOE description

1.4.1 Physical scope of the TOE

The release package for the TOE is composed of software and documents. The TOE software package is in the form of binary compressed file.

All documents and software (including TOE) can be obtained from Huawei support website (<https://cmc-szv.cloudragon.huawei.com/>). For details about how to obtain the document and software, see Section 2.1 of [PRE073].

 **NOTE**

Website (<https://cmc-szv.cloudragon.huawei.com/>) is a HUAWEI internal website, only accessible from Huawei intranet. Users need to use HUAWEI W3 account to log in.

Table 3 describes the physical scope of the TOE, with a description of its different elements.

Table 3 Physical Scope

Software and Documents	Description	Remark
DOPRA SSP V300R005C00SPC123B200 SDK.rar March 2022, SHA256 checksum: 7fb343952f55614728a21ec217d32e106757a5e5930bd0e6adc396cd2ba5df3f	Middleware software package (In the form of binary compressed files). The suffix of the software package is '.rar'	The software package includes the TOE.
Huawei DOPRA SSP V300R005C00SPC123B200 AGD_OPE v0.73.pdf February 2023, SHA256 checksum: 758bb09cee309c4eaf86bfc1eec2f3debc226e394288b25c6114714d84012477	Security management guide.	The security management guide of DOPRA SSP Software.
Huawei DOPRA SSP V300R005C00SPC123B200 AGD_PRE v0.73.pdf March 2023, SHA256 checksum: 3406a247d916720f1394fc3f577f60bf75cb0eb59c6c9fb053fca5be032d34f	Installation Guide.	The installation guide of DOPRA SSP Software.
Huawei DOPRA SSP V300R005C00SPC123B200 API and Configuration Reference.chm V3.0 August 2022, SHA256 checksum: 9e70462503a99a3ed5cc34a72d4ad5ec0ed5e8c9f57c141d5c1d7bf1c6eea788	Interface guide	The interface guide of DOPRA SSP Software.

1.4.2 Logical scope of the TOE

The security features of the TOE are:

1. Auditing

- Records log in the TOE internal memory, when a security event is detected by the TOE,
- Associates timestamp with each security events detected by the TOE,

2. Message Protection

- Monitoring and Alarm
 - When message is damaged, the TOE sends alarm to the APP, and generates an event log,
- Message transmission verification
 - Inserts CRC (for more details about CRC, please refers to Appendix C) in each outgoing message
 - Check the CRC for each incoming message. If the verification fails, the message is discarded, the TOE generates an event log.


3. Memory monitoring

- Monitoring and Alarm
 - When memory is damaged, the TOE sends alarm to the APP, and generates an event log

4. Black Box Protection

- Supports log dumping, when the memory black box area is full

5. Resource Utilization

- Resource specification restriction and partition
 - Creates different message/memory partitions and message token buckets for each module with their own minimum and maximum size setting,
 - Message priority management to ensure that high-priority messages obtain resources first in each module
 - When each message/memory partition usage is exceeding the threshold, the TOE sends alarm to APP, and generates an event log,
 - Each APP process partitions (message and memory) are independent, even if one partition is damaged or insufficient, partitions from another APP process will be not affected.
- Running Overload Protection
 - The message scheduling supports message priority scheduling control. In addition, supports the message token bucket mechanism to implement message flow control.
-  **NOTE**
- **Dumping:** Action to write the content of the memory black box area into a file in the black box file system.
- **APP Process:** A consistent and closely related software organization, consisting of program (APP linked to the TOE) and data structures. One process runs multiple modules.

1.4.3 Evaluated configuration

TOE can run on many types of hardware platform and operating system; the evaluated configuration is as follow:

Table 4 Evaluated configuration

Type	Configuration
Board Type	PC (X86-64bit)
OS layer	Debian GNU/Linux 11 (bullseye)
VM	VMware station 16
VM OS + CPU	suselinux12 + X86_64
VM Hard Disk	2GB
VM RAM	2GB
LIB	DOPRA SSP V300R005C00SPC123B100 SDK.rar Evaluated lib type is suselinux12+x86_64.

2 Conformance claims

2.1 CC Conformance Claim

This ST and the TOE conform to the version of CC as below:

- Part 1: Introduction and general model [Version 3.1 Revision 5](#) [CC1]
- Part 2: Security functional components [Version 3.1 Revision 5](#) [CC2]
- Part 3: Security assurance components [Version 3.1 Revision 5](#) [CC3]

This ST conforms to CC Part 2 conformant.

This ST conforms to CC Part 3 conformant.

This ST is EAL4 conformant as defined in [CC] Part 3, with the assurance level of EAL4 Augmented with [ALC_FLR.1](#).

3 Security Problem Definition

The security problems addressed by the TOE and the operational environment of the TOE are defined in this section. Security problem definition shows the threats that are to be countered by the TOE, its operational environment, or a combination of the two.

This chapter identifies OSP (organizational security policies) as P.OSP and assumptions as A.Assumption.

3.1 Assets

The information assets to be protected are:

3.1.1 Memory partition

- **Memory partition:** memory resource allocated to a TOE instance. The memory resource consists of multiple memory partitions.

The memory partition contains several memory blocks. The structure of a memory block is:

32-bit OS		64-bit OS
8B	Ah = Algorithm head	16B
4B	Hm = Head magic	8B
40B	CS = Call stack	80B
0B	Align padding field	4B
2B	Ln = Alloc file line	2B
2B	Align padding field	2B
4B	Fn = Alloc file name	8B
4B	Tk = Time of alloc	8B
4B	Sz = Memory size	8B
4B	Prv = Forward link	8B
4B	Nxt = Backward link	8B
1B	Opt = Option value	1B
1B	Pt = Partition number	1B
0B	Align padding field	4B
2B	Chk = Checksum value	2B
	UD = User data	
4B	TF = Tail magic	4B

Ah: Algorithm Management Head
Hm: Head magic, including partition and magic words
CS: Call stack when the app requests memory
Ln: File line number to request memory
Fn: File where the code for memory is located
Tk: Time when memory is created
Sz: Size of the requested memory
Prv: Forward pointer of the memory debugging header
Nxt: Backward pointer of the memory debugging header
Opt: debugging options
Pt: The partition where the memory is located
Chk: checksums.
UD: Available memory space for app
TF: End flag of memory data

Note: align padding field is different between 32-bits OS and 64-bits OS.

Remark

Tk use Tick, which counts the total running time after TOE is started, and its unit is 10ms, that is, 100 TICK means that TOE has been running for 1000ms after starting.

The memory block can contain:

- APP data (UD),

- Metadata (Memory management structure data: Header (Ah, Hm, CS, Ln, Fn, Tk, Sz, Prv, Nxt, Opt, Pt, Chk) and TF).

3.1.2 Message partition and UIPC Message

- **Message partition:** specific memory used for APP instance communication. The message resource consists of multiple message partitions.

The message partitions contain several messages. The structure of a message is:

32-bit OS		64-bit OS
4B	Id	4B
4B	St	4B
4B	Sz	4B
4B	CRC	4B
4B	Cid	4B
8B	TA	8B
8B	TH	8B
4B	Tl	4B
16B	CS	32B
12B	BH	12B
64B	IH	64B
4B	PH	4B
20B	Res	20B
32B	CH	32B
	UD	
4B	TF	4B

Id: Fixed identification number of the message header

St: Status that indicates whether the message is damaged.

Sz: Size of the requested message

CRC: CRC check code

Cid: DComponent ID

TA: Time when the app requests message

TH: Time when the TOE handles the message

Tl: Thread ID for processing the message

CS: Call stack when the app requests message

BH: Block Header

IH: UIPC header

PH: PAM (protocal adapter module) header

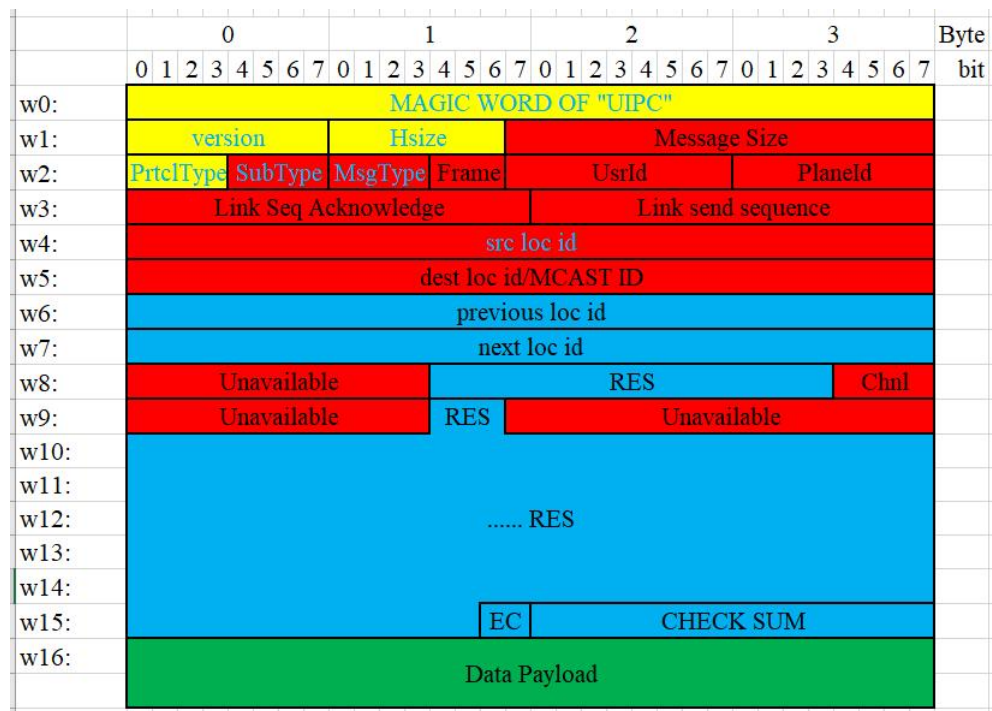
Res: Reserved header
 CH: COMM header
 UD: Available message space for app
 TF: End flag of message data

Remarks:

TA and TH use CPU Tick, which is a high-precision clock, which is generally a multiplier of the CPU's main frequency crystal oscillator, so the accuracy can generally reach the nanosecond level.

A message can contain:

- APP data (UD).
 - Metadata (Message management structure data: Header (Id, St, Sz, CRC, Cid, TA, TH, TI, CS, BH, IH, PH, Res, CH) and TF).
- **UIPC Message:** When TOE sends a message to the receiving DComponent, it needs to convert the message of the partition to the message of UIPC and send it to the receiving DComponent using the UIPC protocol. The following is the format of the general UIPC user data message:



Field	Name	Instruction
WORD-0	MAGIC WORD OF "UIPC"	MAGIC WORD OF "UIPC"
WORD-1	version	UIPC version, the current version number is 1
	Hsize	UIPC header size (Byte)
	Message Size	UIPC message size (Byte)

Field	Name	Instruction
WORD-2	PrtclType	UIPC protocol type
	SubType	Protocol subtype
	MsgType	UIPC message type
	Frame	Indicates message organization
	UsrId	Independent release of user-defined ID
	PlaneId	Plane ID
WORD-3	Link Seq Acknowledge	The response sequence number of the message sent by the link
	Link send sequence	The message sequence number of the message sent by the link
WORD-4	src loc id	ID of the source process of the message
WORD-5	dest loc id/MCAST ID	Message destination receiving process ID/ Multicast ID for sending multicast packets
WORD-6	previous loc id	The last hop process ID of the packet forwarding
WORD-7	next loc id	Next hop process ID for packet forwarding
WORD-8	Unavailable	Random value
	RES	Message reserved field
	Chnl	Plane communication channel number
WORD-9	Unavailable	Random value
	RES	Message reserved field
	Unavailable	Random value
WORD-10~ WORD-14	RES	Message reserved field
WORD-15	EC	Message check type coding
	CHECK SUM	Checksum of the message - CRC
Others	Data payload	Message payload

A UIPC message can contain:

- APP data (Data Payload).
- Metadata (Message management structure data: WORD-0~WORD-15).

3.1.3 Black box and black box file

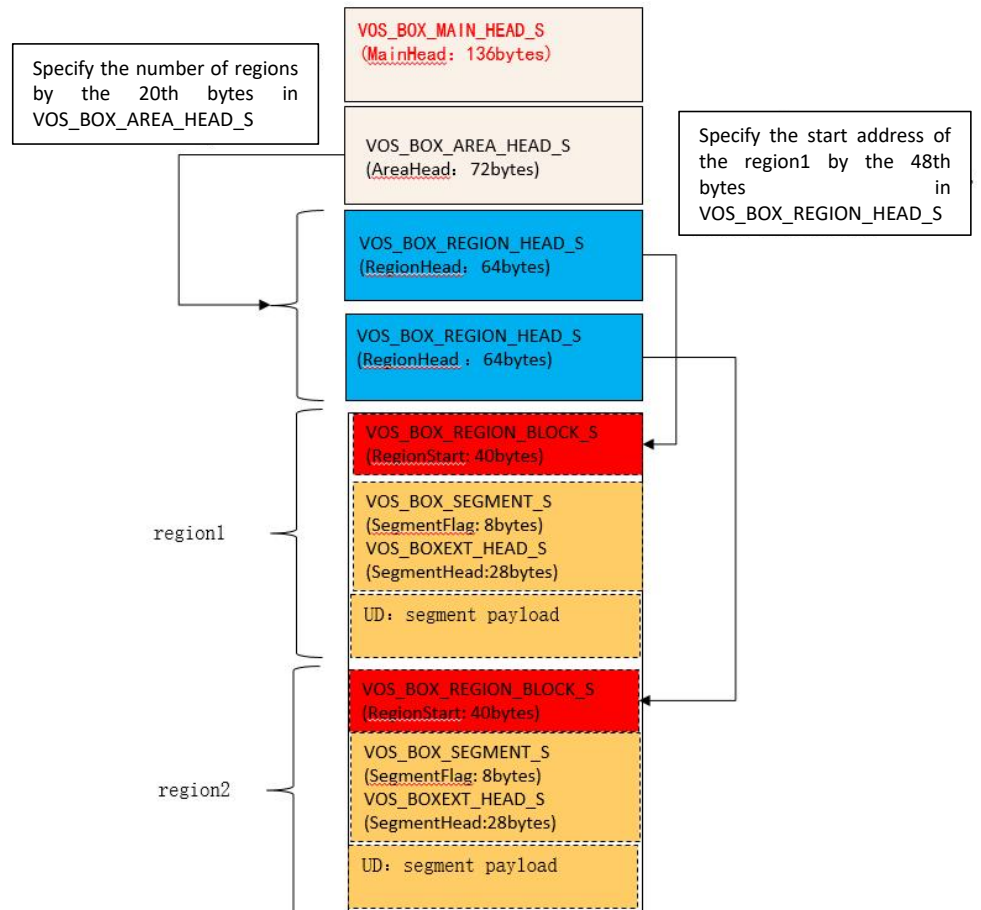
- **Black box:**
The black box is composed of 2 parts: memory and file.

The black box memory consists of **black box areas**. Each black box area consists of multiple **black box regions**, and each black box regions consists of only one **black box segment**, and each black box segment consists of multiple **audit records**.

For a **black box area**, it is a continuous memory space that can contain multiple black box regions. Each black box region is used to store a type of audit records. Generally, each app can apply for a black box region.

For a **black box region**, it has only one black box segment. Since a region has only one segment, when create a region, actually create a segment.

The following figure shows the organization of the **black box** memory:



MainHead: Main control head, which contains device information, version information and the number of areas.
 AreaHead: Area control head. An area is a continuous memory space that can contain multiple regions. Each black box region is used to store a type of audit records. Generally, each app can apply for a black box region.
 RegionHead: Region control head. Specify the start address of the region through the uiOffset field, that is, the first address of VOS_BOX_REGION_BLOCK_S.
 RegionStart: Record the segment length of the region.
 SegmentFlag: Segment identification, inside a fix magic word and some reserve fields.
 SegmentHead: Information about the black box segment. A black box segment can contain multiple records.
 UD: Content of the black box segment, which contains multiple records.

The structure of a black box **segment** is as following:

4B	CRC			
4B	FO			
4B	LO			
4B	TO			
4B	Idx			
4B	RN			
4B	RS			
	UD			

CRC: CRC check code
 FO: Offset of the first record
 LO: Offset of the last record
 TO: Offset of the tail record
 Idx: Serial number
 RN: Number of records written in the segment
 RS: Size of the segment
 UD: Available space for records

The structure of an audit **record** is:

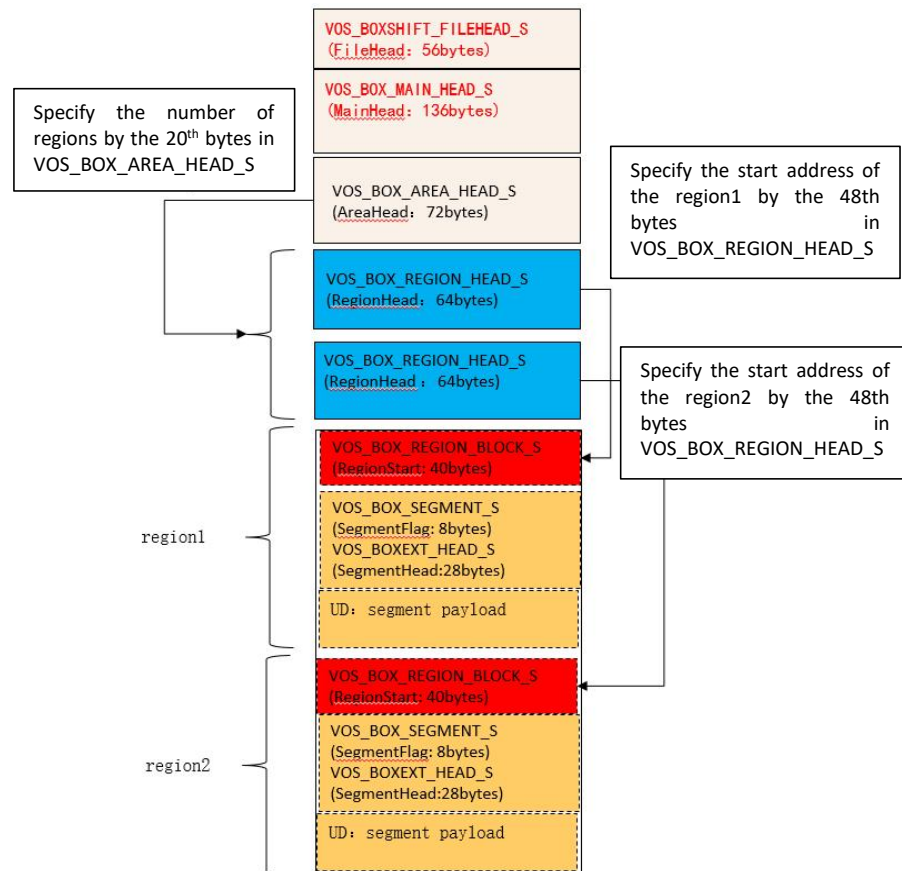
4B	CRC			
4B	ST			
4B	RS			
4B	Idx			
4B	EI			
4B	PO			
4B	NO			
52B	EH			
	UD			

CRC: CRC check code
 ST: Number of system startups
 RS: Size of the record
 Idx: Serial number
 EI: Event id
 PO: Offset from the previous record
 NO: Offset to the next record
 EH: Black box event public header information
 UD: Available space for app

When the black box segment is full, the contents of the black box segment are dumped to the black box file.

The format of the black box file is the same as that of the black box memory, except that FileHead is added to the header of MainHead to save information such as the time of the dump.

The following figure shows the organization of the area and black box segment in the **black box file**:



MainHead: Main control head, which contains device information, version information and the number of areas.
 AreaHead: Area control head. An area is a continuous memory space that can contain multiple regions. Each black box region is used to store a type of audit records. Generally, each app can apply for a black box region.
 RegionHead: Region control head. Specify the start address of the region through the uiOffset field, that is, the first address of VOS_BOX_REGION_BLOCK_S.
 RegionStart: Record the segment length of the region.
 SegmentFlag: Segment identification, inside a fix magic word and some reserve fields.
 SegmentHead: Information about the black box segment. A black box segment can contain multiple records.
 UD: Content of the black box segment, which contains multiple records.

The black box segment records can contain (The storage space can be memory or file):

- Audit records: records of detected security events (UD).
- Metadata (Record management structure data: Header (CRC, ST, RS, Idx, EI, PO, NO, UD)).

3.2 Organizational Security Policies (OSP)

The organizational security policies are listed below.

3.2.1 P.EVENTS

All activities from the TOE shall be recorded and the log shall be dumped in file when the audit trail is full.

3.2.2 P.INTEGRITY

The TOE shall validate CRC in each incoming message; and provide CRC in each outgoing message.

3.2.3 P.ALARM

The TOE shall generate an alarm for all the relevant events

3.2.4 P.RESOURCE.UTILIZATION

The TOE shall set-up parameters for message/memory partitions, segmentation mechanisms.

3.3 Assumptions

The specific conditions below are assumed to exist in a TOE environment.

3.3.1 A.PHYSICAL

It is assumed that the IT environment provides the TOE with appropriate physical security, in line with the value of the IT assets protected by the TOE. This assumption also ensures that only authorized APP can be linked to the TOE.

It is assumed that the authentication of the users is managed by the IT environment and only authorized users can have access to the TOE configuration file.

3.3.2 A.NOEVIL

It is assumed that the APP using the TOE are developed by programmers who are not hostile, careless or wilfully negligent observing the instructions provided by the TOE guidance documentation.

3.3.3 A.INSTALL

It is assumed that those responsible for the TOE must establish and implement procedures to ensure that the hardware, software and firmware components that comprise the system are distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE.

3.3.4 A.RESOURCE

It is assumed that when the TOE is started, the IT environment can allocate proper running resources to the TOE to ensure its normal execution.

It is assumed that when an app user uses a tool to parse a black box file, it will verify the CRC at the beginning of the black box segment in the file to ensure the integrity of the black box file.

3.3.5 A.OS

It is assumed that the OS has no vulnerabilities, and the permissions are adequately set-up in a way to protect the files containing the dumped logs and the configuration file. The configuration file is assumed to be well formed. It is also assumed that the execution platform is trustworthy and that a user cannot interact directly with the hardware through the assembly language. Additionally, it is assumed that the authorized APP to which the TOE is linked does not constitute an attack path since the APP will not present malicious behaviour or act as an attacker.

3.3.6 A.TIME

It is assumed that clock used by the NTP client is reliable. It is also assumed the interrupt of hardware is reliable.

4 Security Objectives

The security objectives are divided into two solutions. The security objectives for the TOE and the security objectives for the operational environment. It reflects that these solutions are provided by two different entities: the TOE, and the supporting environment.

4.1 Security Objectives for the TOE

Table 5 Security Objectives for the TOE

Security Objective	Description
O.AUDITING	<p>The TOE must:</p> <ul style="list-style-type: none"> Record log in the black box memory, when a security event is detected by the TOE Associate timestamp with each detected security events detected by the TOE
O.MSG.PROTECTION	<p>The TOE must provide:</p> <ul style="list-style-type: none"> Checking the CRC on each incoming message to detect message tampering Adding CRC in each outgoing message Message damage checking mechanism to detect message damage (Invalid overwriting). Generate an event log when message damage is detected. A severity level (INFO, ERROR, WARNING or EXCEPT is added to this event. An alarm is sent to the APP when message damage is detected.
O.MEM.MONITORING	<p>The TOE must provide:</p> <ul style="list-style-type: none"> memory damage checking mechanisms to detect the memory damage (Invalid overwriting),

Security Objective	Description
	<ul style="list-style-type: none"> generate an event log when memory damage is detected. A severity level (INFO, ERROR, WARNING or EXECPT) is added to this event and an alarm is sent to the APP.
O.BB.PROTECTION	<p>The TOE must provide:</p> <ul style="list-style-type: none"> log dumping, when the black box memory is full.
O.RESOURCE.UUTILIZATION	<p>The TOE must provide:</p> <ul style="list-style-type: none"> creation of different message/memory partitions for each module. Creation of several message token bucket, for each module, each memory partition has its own minimum size setting. Each message partition and message token bucket has its own maximum size setting. message priority to ensure that high-priority APP process obtain resources first. The partition-based message overload monitoring mechanism prevents an APP process from using too many message resources and affecting the running of other APP processes. The partition-based memory overload monitoring mechanism prevents an APP process from using too many memory resources and affecting the running of other APP processes. segmentation mechanisms to ensure that different APP process using different segments and do not affect each other. message token bucket mechanism to implement message flow control. When a communication link is established between a sender DComponent and a receiver DComponent, the system creates a message token bucket for the sender DComponent. A certain number of message tokens are stored in the bucket. When sending a message, the sender needs to apply for a token. After the peer end to process the message, the sender returns the token to the sender. When tokens are used up, the sender enters the flow control state and stops sending messages to the recipient.

4.2 Security Objectives for the Operational Environment

The operational environment of the TOE implements technical and procedural measures to assist the TOE in correctly providing its security functionality (which is defined by the security objectives for the TOE).

Table 6 Security Objectives for the Operational Environment

Security Objective	Description
OE.PHYSICAL	Those responsible for the TOE must ensure that those parts of the TOE critical to enforcement of the security policy are protected from physical attack that might compromise IT security objectives. The protection must be consistent with the value of the IT assets protected by the TOE. All the TOE instances are in one

Security Objective	Description
	frame which is a trust zone.
OE.TIME	The timestamp is provided by the clock. The clock is synchronized periodically through NTP. During the synchronization, the clock is maintained by hardware interruption. The clock used by the NTP client, and the hardware are reliable.
OE.ADMIN	Those responsible for the TOE are competent and trustworthy individuals, capable of managing the TOE and the security of the information it contains.
OE.INSTALL	Those responsible for the TOE must establish and implement procedures to ensure that the hardware, software and firmware components that comprise the system are distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE.
OE.RESOURCE	When the system is started, the system user can allocate proper running resources to the DOPRA SSP to ensure the normal running of the SSP functional components. When an app user uses a tool to parse a black box file, it will verify the CRC at the beginning of the black box segment in the file to ensure the integrity of the black box file.
OE.OS	The OS where the TOE is installed, has no vulnerabilities, and the permissions are adequately set-up in a way to protect the files containing the dumped logs and the configuration file that is well formed. The execution platform is trustworthy and users cannot interact directly with the hardware through the assembly language. Additionally, the authorized APP to which the TOE is linked does not constitute an attack path since the APP will not present malicious behaviour or act as an attacker.

4.3 Security Objectives rationale

4.3.1 Coverage

Table 7 Security Objectives coverage

	O.AUDITING	O.MSG.PROTECTION	O.MEM.MONITORING	O.BB.PROTECTION	O.RESOURCE.UUTILIZATION	OE.PHYSICAL	OE.TIME	OE.ADMIN	OE.INSTALL	OE.RESOURCE	OE.OS
P.EVENTS	X	X	X	X	X						

	O.AUDITING	O.MSG.PROTECTION	O.MEM.MONITORING	O.BB.PROTECTION	O.RESOURCE.UTILIZATION	OE.PHYSICAL	OE.TIME	OE.ADMIN	OE.INSTALL	OE.RESOURCE	OE.OS
P.INTERGRITY		X									
P.ALARM		X	X								
P.RESOURCE.UTILIZATION					X						
A.PHYSICAL	X	X	X	X	X	X					
A.NOEVIL								X			
A.INSTALL									X		
A.RESOURCE										X	
A.TIME							X				
A.OS											X

Table 8 Security Objectives rationale

Assumption/OSP	Rationale for objectives
P.EVENTS	<p>O.AUDITING ensures that the logs are recorded in black box memory when a security event is detected.</p> <p>O.MSG.PROTECTION ensures message verification fails and message corruption detected, shall be record in the log.</p> <p>O.MEM.MONITORING ensures memory corruption detected shall be record in the log.</p> <p>O.BB.PROTECTION ensures that the log shall be dumped in file.</p> <p>O.RESOURCES.UTILIZATION ensures memory overload, messages overload detected, shall be recorded in the log.</p>
P.INTERGRITY	<p>O.MSG.PROTECTION ensures that there are CRC in each outgoing message, and check the CRC on each incoming message.</p>
P.ALARM	<p>O.MSG.PROTECTION ensure that an alarm is generated when message corruption is detected.</p> <p>O.MEM.MONITORING ensure that an alarm is generated when memory corruption is detected.</p>
P.RESOURCE.UTILIZATION	<p>O.RESOURCES.UTILIZATION</p>

Assumption/OSP	Rationale for objectives
	<p>ensure that there is partitions function for memory and message,</p> <p>ensure different app shall not affect each other in the black box,</p> <p>ensure the message flow control,</p> <p>ensure to monitor the memory overload, message overload,</p> <p>ensure that high-priority APP process obtain resources first.</p>
A.PHYSICAL	<p>OE.PHYSICAL ensures that those parts of the TOE critical to enforcement of the security policy are protected from physical attack.</p> <p>O.AUDITING, O.MSG.PROTECTION, O.MEM.MONITORING, O.BB.PROTECTION and O.RESOURCEUTILIZATION can perform as intended given the assumption from A.PHYSICAL that the TOE is physically unharmed.</p>
A.NOEVIL	<p>OE.ADMIN ensures that responsible for the TOE are competent and trustworthy individuals, capable of managing the TOE and the security of the information it contains.</p>
A.INSTALL	<p>OE.INSTALL ensures that responsible for the TOE must establish and implement procedures to ensure that the hardware, software and firmware components that comprise the system are distributed, installed and configured in a secure manner supporting the security mechanisms provided by the TOE.</p>
A.RESOURCE	<p>OE.RECOURSE ensures that when the system is started, the system user can allocate proper running resources to the DOPRA SSP to ensure the normal running of the SSP functional components.</p>
A.TIME	<p>OE.TIME ensures the clock is synchronized periodically through NTP. During the synchronization, the clock is maintained by hardware interruption The clock used by the NTP client and the hardware are reliable.</p>
A.OS	<p>OE.OS ensures that OS has no vulnerabilities, and the permissions are adequately set-up in a way to protect the files containing the dumped logs. Additionally, OE.OS ensures that the authorized APP to which the TOE is linked does not constitute an attack path since the APP will not present malicious behaviour or act as an attacker.</p>

5 Extended Components Definition

This ST defines no extended security functional components. All security functional requirements stated for the TOE are stated in [CC2].

6 Security Requirements for the TOE

This section provides functional and assurance requirements that satisfied by the TOE. These requirements consist of functional components from Part 2 of the CC and an Evaluation Assurance Level (EAL) containing assurance components from Part 3 of the CC.

6.1 Conventions

The following conventions are used for the completion of operations:

- ~~Strikethrough~~ indicates text removed as a refinement
- (Underlined text in parentheses) indicates additional text provided as a refinement.
- **[Bold text]** indicates the completion of an assignment.
- ***[Italicized and bold text]*** indicates the completion of a selection.
- Iteration/N indicates an element of the iteration, where N is the iteration element name.

6.2 Security Functional Requirements

The functional security requirements for the TOE consist of the following components from Part 2 of the CC, summarized in the following table.

Table 9 Security Functional Requirements

	Functional Requirements
FAU_GEN.1	Audit data generation
FAU_SAA.1	Potential violation analysis
FAU_ARP.1	Security alarms
FAU_STG.4	Prevention of audit data loss
FRU_PRS.1	Limited priority of service
FRU_RSA.2	Minimum and maximum quotas
FDP_SDI.2	Prevent data user modification and action
FDP_UIT.1	Data exchange integrity
FDP_IFC.1	Subset information flow control
FDP_IFF.1	Simple security attributes

6.2.1 Security Audit (FAU)

6.2.1.1 FAU_GEN.1 Audit data generation

FAU_GEN.1.1: The TSF shall be able to generate an audit record of the following auditable events:

- a. Start-up and shutdown of the audit functions;
- b. All auditable events for the [*selection: not specified*] level of audit; and
- c. [*assignment:*
 - i. **Memory overload;**
 - ii. **Message overload;**
 - iii. **Message tampered;**
 - iv. **Memory damaged;**
 - v. **Message damaged;**].

FAU_GEN.1.2: The TSF shall record within each audit record at least the following information:

- a. Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; and
- b. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [*assignment: TOE Instance ID*].

6.2.1.2 FAU_SAA.1 Potential violation analysis

FAU_SAA.1.1: The TSF shall be able to apply a set of rules in monitoring the audited events and based upon these rules indicate a potential violation of the enforcement of the SFRs.

FAU_SAA.1.2: The TSF shall enforce the following rules for monitoring audited events:

- a. Accumulation or combination of [*assignment: memory damaged, message damaged*] known to indicate a potential security violation;
- b. [*assignment: no other rules*].

6.2.1.3 FAU_ARP.1 Security alarms

FAU_ARP.1.1: The TSF shall [*assignment: record event log in memory with associated timestamp, record event log in black box, send an alarm to the APP*] upon detection of a potential security violation.

6.2.1.4 FAU_STG.4 Prevention of audit data loss

FAU_STG.4.1: The TSF shall [*selection: overwrite the oldest stored audit records*] and [*assignment: dump the audit records from memory to file*] if the audit trail is full.

6.2.2 Resource Utilization (FRU)

6.2.2.1 FRU_PRS.1 Limited priority of service

FRU_PRS.1.1: The TSF shall assign a priority to each subject in the TSF.

FRU_PRS.1.2: The TSF shall ensure that each access to [*assignment: Message Scheduling*] shall be mediated on the basis of the subjects assigned priority.

6.2.2.2 FRU_RSA.2 Minimum and maximum quota

FRU_RSA.2.1: The TSF shall enforce maximum quotas of the following resources [assignment: memory partition] that [selection: *Individual user*] can use [selection: *simultaneously*].

Application note: Individual user refer to APP only.

FRU_RSA.2.2: The TSF shall ensure the provision of minimum quantity of each [assignment: message partition, message token bucket] that is available for [selection: *Individual user*] to use [selection: *simultaneously*].

Application note: Individual user refer to APP only.

6.2.3 User Data Protection (FDP)

6.2.3.1 FDP_SDI.2 Stored data integrity monitoring and action

FDP_SDI.2.1 The TSF shall monitor user data stored in containers controlled by the TSF for [assignment: i. **Memory damaged.**]

On all objects, based on the following attributes: [assignment:

- i. **Checksum value of the memory block and the tail magic at the end of the memory block.**]

FDP_SDI.2.2 Upon detection of a data integrity error, the TSF shall [assignment: **generate an event log and send an alarm to APP**].

Application note: refer to section 3.1.1 for details about Checksum value and tail magic.

6.2.3.2 FDP_UIT.1 Data exchange integrity

FDP_UIT.1.1 The TSF shall enforce the [assignment: **message flow control policy**] to [selection: *transmit, receive*] user data in a manner protected from [selection: *modification, deletion, insertion*] errors.

FDP_UIT.1.2 The TSF shall be able to determine on receipt of user data, whether [selection: *modification, deletion, insertion*] has occurred.

6.2.3.3 FDP_IFC.1 Subset information flow control

FDP_IFC.1.1 The TSF shall enforce the [assignment: **message flow control policy**] on [assignment:

Subjects: TOEs

Information: message

Operation: sending and receiving message between different TOEs].

6.2.3.4 FDP_IFF.1 Simple security attributes

FDP_IFF.1.1 The TSF shall enforce the [assignment: **message flow control policy**] based on the following types of subject and information security attributes:

[assignment:

Subjects: TOEs,

Information: message

Security attribute: Process ID for the TOE, CRC and destination process ID for message]

FDP_IFF.1.2 The TSF shall permit an information flow between a controlled subject and controlled information via a controlled operation if the following rules hold: [assignment: **If the Process ID for the receiver TOE matches the destination process ID of the received message AND the CRC value in the**

received message matches the CRC value calculated by the receiver TOE, the message flow is allowed].

FDP_IFF.1.3 The TSF shall enforce the [assignment: none].

FDP_IFF.1.4 The TSF shall explicitly authorise an information flow based on the following rules: [assignment: message sending is always allowed].

FDP_IFF.1.5 The TSF shall explicitly deny an information flow based on the following rules: [assignment: none].

6.3 Security Requirements Dependency Rationale

The security functional requirements in this Security Target do not introduce dependencies on any security assurance requirement; neither do the security assurance requirements in this Security Target introduce dependencies on any security functional requirement.

The following table demonstrates the dependencies of SFRs modelled in CC Part 2 and how the SFRs for the TOE resolve those dependencies:

Table 10 Dependencies between TOE Security Functional Requirements

Security Functional Requirement	Dependencies	Resolution
FAU_GEN.1	FPT_STM.1	Met by the environment. Although FPT_STM.1 is not included, it is stated that OE.TIME provides reliable timestamps to the TOE.
FAU_SAA.1	FAU_GEN.1	FAU_GEN.1
FAU_ARP.1	FAU_SAA.1	FAU_SAA.1
FAU_STG.4	FAU_STG.1	Met by Operational Environment. Although FAU_STG.1 is not included, it is stated in OE.OS that the OS would provide log protection by its access control.
FRU_PRS.1	NA	NA
FRU_RSA.2	NA	NA
FDP_SDI.2	NA	NA
FDP_UIT.1	[FDP_ACC.1, or FDP_IFC.1] [FTP_ITC.1, or FTP_TRP.1]	FDP_IFC.1 [FTP_ITC.1, or FTP_TRP.1] is met by the environment. OE.PHYSICAL ensures that the TOE is well-protected physically. Therefore, only trusted device can communicate with the TOE.
FDP_IFC.1	FDP_IFF.1	FDP_IFF.1
FDP_IFF.1	FDP_IFC.1	FDP_IFC.1

Security Functional Requirement	Dependencies	Resolution
	FMT_MSA.3	FMT_MSA.3 Although FMT_MSA.3 is not included, the TOE does not maintain any role and does not authorise any role. Moreover, providing default values for the Security attributes does not serve any purpose in this particular case.

6.4 Security Functional Requirements Rationale

6.4.1 Coverage

The following table provides a mapping of SFRs to the security objectives, showing that each security functional requirement addresses at least one security objective.

The following rationale provides justification for each security objective for the TOE, showing that all security objectives are addressed, and the security functional requirements are suitable to meet and achieve the security objectives.

Table 11 Security Objectives Coverage

	O.AUDITING	O.MSG.PROTECTION	O.MEM.MONITORING	O.BB.PROTECTION	O.RESOURCE.UUTILIZATION
FAU_GEN.1	X	X	X		X
FAU_SAA.1		X	X		
FAU_ARP.1	X	X	X		
FAU_STG.4				X	
FRU_PRS.1					X
FRU_RSA.2					X
FDP_SDI.2			X		
FDP_UIT.1		X			
FDP_IFC.1		X			
FDP_IFF.1		X			

6.4.2 Sufficiency

The following rationale provides justification for each security objective for the TOE, showing that the security functional requirements are suitable.

Table 12 SFR sufficiency analysis

Security objectives	Rationale
O.AUDITING	<p>FAU_ARP.1 ensures that security event log recorded in black box, and an alarm is sent to the APP. FAU_GEN.1 ensures a security event log will have an associated timestamp.</p>
O.MSG.PROTECTION	<p>Message damage is detected according to FAU_SAA.1.</p> <p>FAU_GEN.1 ensures a security event log will be created when message tampering and message damaged are detected.</p> <p>FAU_ARP.1 ensures that an alarm is sent to the APP when a message damaged is detected.</p> <p>FDP_UIT.1 ensures that message is not damaged when a message is transmitted.</p> <p>FDP_IFC.1 establishes the subjects, information and operations that are controlled by the message flow control policy for integrity checking.</p> <p>FDP_IFF.1 ensures that security attributes of message are implemented, and message flow control policy are enforced.</p>
O.MEM.MONITORING	<p>Memory damage is detected according to FDP_SDI.2 and FAU_SAA.1.</p> <p>FAU_GEN.1 ensures a security event log will be created when memory damaged is detected.</p> <p>FAU_ARP.1 ensures that an alarm is sent to the APP when a memory damaged is detected.</p>
O.BB.PROTECTION	<p>FAU_STG.4 ensures that when the black box memory is full the event logs will be dump to file or it will overwrite the oldest stored audit records.</p>
O.RESOURCE.UTILIZATION	<p>FAU_GEN.1 ensures a security event log will be created when message overload, memory overload are detected.</p> <p>FRU_PRS.1 ensures that high-priority APP process obtain resources.</p> <p>FRU_RSA.2 ensures that each memory partition has its own minimum size setting. Each message partition and message token bucket has its own maximum size setting.</p>

6.5 Security Assurance Requirements

The security assurance requirements for the TOE are the Evaluation Assurance Level 4 components as specified in [CC3], augmented with ALC_FLR.1. No operations are applied to the assurance components.

Table 13 Security Assurance Requirements

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level
Development	ADV_ARC	1
	ADV_FSP	4
	ADV_IMP	1
	ADV_TDS	3
Guidance documents	AGD_OPE	1
	AGD_PRE	1
Life-cycle support	ALC_CMC	4
	ALC_CMS	4
	ALC_DEL	1
	ALC_DVS	1
	ALC_FLR	1
	ALC_LCD	1
	ALC_TAT	1
Security Target evaluation	ASE_CCL	1
	ASE_ECD	1
	ASE_INT	1
	ASE_OBJ	2
	ASE_REQ	2
	ASE_SPD	1
	ASE_TSS	1
Tests	ATE_COV	2
	ATE_DPT	1
	ATE_FUN	1
	ATE_IND	2
Vulnerability assessment	AVA_VAN	3

6.6 Security Assurance Requirements Rationale

The EAL4 was chosen to permit a developer to gain maximum assurance from positive security engineering based on good commercial development practices which, although rigorous, do not require substantial specialist knowledge, skills, and other resources.

EAL4 is applicable in those circumstances where developers or users require a moderate to high level of independently assured security in conventional commodity TOEs and are prepared to bear sensitive security specific engineering costs.

EAL4 is in line with the level of threats that the user of the TOE will encounter.

The augmentation of ALC_FLR.1 was chosen to give greater assurance of the developer's on-going flaw remediation.

7 TOE Summary Specification

7.1 Auditing

1. All audit records are recorded in the black box segment created by the log module. (FAU_ARP.1)
2. When the TOE is running, logs are recorded once the log point is triggered. Based on the impact on the system, logs are classified into the following alarm severity: INFO, ERROR, WARNING, and EXECPT. (FAU_ARP.1)
3. When recording the log, some additional information is added to the log, such as the module ID, recording time, call stack and other information of the module that generated the log, which is convenient to track the log. (FAU_GEN.1)
4. Module used by the TOE for identity the event. There are three levels for event security which are error, warning and info. (FAU_GEN.1)

7.2 Message Protection

1. When a message is sent, the CRC is added to the protocol header of the message and the CRC is verified upon receiving the message. The number of messages with failed CRC check are recorded in audit log. (FAU_GEN.1, FDP_UIT.1, FDP_IFC.1, FDP_IFF.1)
2. The message damage detection function is provided to verify the CRC at the beginning and end of the message (FDP_UIT.1, FDP_IFC.1, FDP_IFF.1, FAU_SAA.1). A log is generated when a message corruption is detected (FAU_GEN.1) and an alarm is sent to the APP by call-back function (FAU_ARP.1).

7.3 Memory monitoring

1. The memory damage detection function is provided to verify the identification number or CRC code at the beginning of the memory and the identification number at the end of memory (FDP_SDI.2). An event log is generated when a memory corruption is detected (FAU_GEN.1 and FDP_SDI.2) and an alarm is sent to the APP by call-back function (FAU_ARP.1, FDP_SDI.2 and FAU_SAA.1).

7.4 Black Box Protection

1. Provides black box data dump function. (FAU_STG.4)

The black box segment supports acyclic overwrite mode and cyclic overwrite mode.

For the acyclic overwrite mode, when the black box segment space is full, the content of the black box segment and the dump time will be recorded in a file. The content of the black box segment will then be removed. And the new entries will be recorded in the black box segment space again.

For the cyclic overwrite mode, if the APP turns on the dump function, the information dump will be automatically performed before the operating system restarts. In this mode, when a black box segment space in the black box is full, the black box segment will be automatically overwritten the oldest entries.

7.5 Resource Utilization

1. Provides partitioning mechanism for memory.

The TOE provides the partitioning function. Several memory partitions (Defined by API) can be created with a minimum size (FRU_RSA.2).

When the memory reaches the total allocated memory size, but the TOE returns a failure.

2. Provides overload monitoring for memory

The TOE provides partition-based memory overload monitoring the memory. Each partition has its own overload threshold and recovery threshold. When the use of memory reaches the overload threshold, the partition enters the overload state, and a log entry is recorded (FAU_GEN.1).

3. Provides partitioning mechanism for messages.

The TOE provides the partitioning function. Several message partitions (Defined by API) can be created with a maximum size.

When the size of the messages in the partition reaches the total allocated memory size, the TOE applies for a large memory block from the operating system and divides the memory block into a certain number of messages. A maximum number of messages is defined for each partition, and when the maximum number of managed messages is reached, the messages exhaustion does not apply memory from the operating system but returns a failure. (FRU_RSA.2).

4. Provides overload monitoring for messages

The TOE provides partition-based message overload monitoring the messages. Each partition has its own overload threshold and recovery threshold. When the number of used messages reaches the overload threshold, the partition enters the overload state, and a log entry is recorded (FAU_GEN.1).

5. Provides the message token bucket mechanism to implement message flow control (FRU_RSA.2).

When a communication link is established between a sender DComponent and a receiver DComponent, the system creates a message token bucket for the sender DComponent. The bucket stores a certain number of message tokens. The sender needs to apply for a token when sending a message. After the message is received, the receiver returns the token to the sender. When all the tokens are used, the sender enters the flow control state

and stops sending messages to the recipient. When the sender receives the tokens returned by the receiver, the flow control state is stopped.

6. Provide two levels of scheduling priority. (FRU_PRS.1)

The DComponent scheduling instance supports two priority levels, the scheduling thread preferentially processes the high priority messages, and then processes the low priority messages. The messages that ensure the operation of the system are high-priority messages, such as flow control messages. Other messages used for communication between DComponents are low priority messages.

A

Acronyms and Abbreviations

APP	Customer Application Service
BBU	Base Band Unit
BS	Base station
CC	Common Criteria
CPU	Central Processing Unit
CRC	cyclic redundancy check
DComponent	DComponent is the abbreviation of DOPRA component, the DComponents communicate through messages, and different DComponents provide different services.
DOPRA	Distributed SPP, Object-oriented Programmable Real-time Architecture. DOPRA is the abbreviation of Distributed Object-oriented Programmable Realtime Architecture. It helps to accommodate the difference of upper-layer OS, hardware, network, and system scale.
eNodeB	LTE base station
gNodeB	5G base station
SFR	security functional requirements
NE	network element
NIC	Network Interface Controller
NVRAM	Non-Volatile Random Access Memory
RAM	Random Access Memory
NSA	Non-Standalone,5G only support data related service, use 4G for non-data duties
O&M	operation and maintenance
OMCH	Operation & Maintenance channel
OS	operating system
ST	Security Target
S1-U	The S1-U interface is used to transfer user plane data between gNodeB and S-GW

S-GW	Serving Gateway
TOE	Target of evaluation
TSF	TOE Security Functionality
UBBP	The 5G Baseband Processing and radio Interface Unit, whose purpose is to provide an interface between BBU and Radio Remote Unit (RRU)/ Active Antenna Unit (AAU)
UMPT	The Main Processes and Transmission unit, which is the main board of BBU
X2	The X2 interface is used to transmit the control plane and user plane traffic when the gNodeB works with the eNodeB in the NSA mode
W3	The W3 website is Huawei's unified work platform. It is Huawei's internal portal.

B Technical References

CC1	Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, version 3.1, revision 5, April 2017, ref. CCMB-2017- 04-001
CC2	Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components, version 3.1, revision 5, April 2017, ref. CCMB-2017- 04-002
CC3	Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, version 3.1, revision 5, April 2017, ref. CCMB-2017- 04-003
PRE07	Huawei DOPRA SSP V300R005C00SPC123200 AGD_PRE v0.7, 2022-07-15

C Appendix: Technical of CRC

C.1 Memory Partition CRC

The memory CRC is calculated as follows (assume that the value to be calculated is A):

- Step 1: Pre-set a 16-bit value 0x3a29 as the initial value. The value is marked as an usChecksum.
- Step 2: Perform the XOR operation on the lower two bytes of A and the usChecksum, save the result to the usChecksum, and shift A 16 bits to the right and save the result to A.
- Step 3: Repeat step 3 until the values of A are all moved. The value of the usChecksum is the CRC value of the memory.

The following is the calculation range of the memory checksum, including section 1 and section 2.

32-bit OS		64-bit OS	
8B	Ah = Algorithm head	16B	
4B	Hm = Head magic	8B	
40B	CS = Call stack	80B	← pusSection1Start
0B	Align padding field	4B	} Section 1 range
2B	Ln = Alloc file line	2B	
2B	Align padding field	2B	
4B	Fn = Alloc file name	8B	
4B	Tk = Time of alloc	8B	
4B	Sz = Memory size	8B	← pusSection1End
4B	Prv = Forward link	8B	
4B	Nxt = Backward link	8B	
1B	Opt = Option value	1B	← pusSection2Start
1B	Pt = Partition number	1B	} Section 2 range
0B	Align padding field	4B	
2B	Chk = Checksum value	2B	← pUserAddr
	UD = User data		
4B	TF = Tail magic	4B	

Note: align padding field is different between 32-bits OS and 64-bits OS. In addition, the Align padding field in front of Ah, Hm, Prv, Nxt and Chk do not participate in the CRC calculation, so these fields are damaged and memory damage cannot be detected.

The following is the detailed calculation process as example:

pre-condition of this example

user data (UD) address of memory (pUserAddr): 0x7fff941e2ca8

system configuration: host-order, little-endian, 64-bits

Red: the range of the CRC calculation

Blue: calculated CRC value

```

0x7fff941e2c18: 0xb00aa09d 0x00000000 0x01f7340d 0x00000000
0x7fff941e2c28: 0xffffd94f 0x00007fff 0x0000105b 0x000026e
0x7fff941e2c38: 0xdeaddead 0x00000000 0xdeaddead 0x00000000
0x7fff941e2c48: 0xdeaddead 0x00000000 0xdeaddead 0x00000000
0x7fff941e2c58: 0xdeaddead 0x00000000 0xdeaddead 0x00000000
0x7fff941e2c68: 0xdeaddead 0x00000000 0x00000000 0x0000297a
0x7fff941e2c78: 0x02839600 0x00000000 0x00000041 0x00000000
0x7fff941e2c88: 0x00000080 0x00000000 0xe6b3cfa8 0x00007fff
0x7fff941e2c98: 0xe6b3cfa8 0x00007fff 0x00000907 0xf0960000
0x7fff941e2ca8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2cb8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2cc8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2cd8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2ce8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2cf8: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fff941e2d08: 0x00000000 0x00000000 0x00000000 0x00000000

```

Data value of each element in memory partition:

Section Number	Title	Parse	Value	Remark
-	Hm	Head magic, including partition and magic words	0xb00aa09d 0x00000000	
1	CS	Call stack when the app requests memory	0x01f7340d 0x00000000 0xffffd94f 0x00007fff 0x0000105b 0x000026e 0xdeaddead 0x00000000 0xdeaddead 0x00000000 0xdeaddead 0x00000000 0xdeaddead 0x00000000 0xdeaddead 0x00000000 0xdeaddead 0x00000000 0xdeaddead 0x00000000	
1	Align	Align Padding Field	0x00000000	
1	Ln	File line number to request memory	0x297a	Lower 2 bytes of 0x0000297a
1	Align	Align Padding Field	0x0000	High 2 bytes of 0x0000297a
1	Fn	File where the code for memory is located	0x02839600 0x00000000	
1	Tk	Time when memory is created	0x00000041 0x00000000	

Section Number	Title	Parse	Value	Remark
1	Sz	Size of the requested memory	0x00000080 0x00000000	
-	Prv	Forward pointer of the memory debugging header	0xe6b3cfa8 0x00007fff	
-	Nxt	Backward pointer of the memory debugging header	0xe6b3cfa8 0x00007fff	
2	Opt	debugging options	0x07	Lowest byte of 0x00000907
2	Pt	The partition where the memory is located	0x09	Second-Last Byte of 0x00000907
-	Align	Align Padding Field	0x0000 0x0000	High 2 bytes of 0x00000907 low 2 bytes of 0xf0960000
-	Chk	checksums	0xf096	High 2 bytes of 0xf0960000

Memory:

A = address of CS

initial usChecksum (constant value): 0x3a29

Calculated CRC: 0x6a86

Calculated result for each step:

Step 1: usChecksum: 0x3a29

Step 2: Perform the XOR operation on the lower two bytes of A 0x340d and the usChecksum (input) 0x3a29

Save the result to the usChecksum (output) 0x0e24.

Shift A 16 bits to the right and save the result to A 0x01f7.

Step 3: Repeat step 2 until the values of A are all moved. The value of the usChecksum is the CRC value of the memory.

The following is the intermediate data calculated at each step:

step	loop	lower two bytes of A	Input usChecksum	Output usChecksum
Step 1	NA	NA	0x3a29	0x3a29
Step 2-3	0	0x340d	0x3a29	0x0e24
	1	0x01f7	0x0e24	0xfd3
	2	0x0000	0xfd3	0xfd3
	3	0x0000	0xfd3	0xfd3
	4	0xd94f	0xfd3	0xd69c
	5	0xffff	0xd69c	0x2963
	6	0x7fff	0x2963	0x569c
	7	0x0000	0x569c	0x569c

8	0x105b	0x569c	0x46c7
9	0x0000	0x46c7	0x46c7
10	0x026e	0x46c7	0x44a9
11	0x0000	0x44a9	0x44a9
12	0xdead	0x44a9	0x9a04
13	0xdead	0x9a04	0x44a9
14	0x0000	0x44a9	0x44a9
15	0x0000	0x44a9	0x44a9
16	0xdead	0x44a9	0x9a04
17	0xdead	0x9a04	0x44a9
18	0x0000	0x44a9	0x44a9
19	0x0000	0x44a9	0x44a9
20	0xdead	0x44a9	0x9a04
21	0xdead	0x9a04	0x44a9
22	0x0000	0x44a9	0x44a9
23	0x0000	0x44a9	0x44a9
24	0xdead	0x44a9	0x9a04
25	0xdead	0x9a04	0x44a9
26	0x0000	0x44a9	0x44a9
27	0x0000	0x44a9	0x44a9
28	0xdead	0x44a9	0x9a04
29	0xdead	0x9a04	0x44a9
30	0x0000	0x44a9	0x44a9
31	0x0000	0x44a9	0x44a9
32	0xdead	0x44a9	0x9a04
33	0xdead	0x9a04	0x44a9
34	0x0000	0x44a9	0x44a9
35	0x0000	0x44a9	0x44a9
36	0xdead	0x44a9	0x9a04
37	0xdead	0x9a04	0x44a9
38	0x0000	0x44a9	0x44a9
39	0x0000	0x44a9	0x44a9
40	0x0000	0x44a9	0x44a9
41	0x0000	0x44a9	0x44a9
42	0x297a	0x44a9	0x6dd3
43	0x0000	0x6dd3	0x6dd3
44	0x9600	0x6dd3	0xfbd3
45	0x0283	0xfbd3	0xf950
46	0x0000	0xf950	0xf950
47	0x0000	0xf950	0xf950
48	0x0000	0xf950	0xf950
49	0x0041	0xf950	0xf911

50	0x0000	0xf911	0xf911
51	0x0000	0xf911	0xf911
52	0x0080	0xf991	0xf991
53	0x0000	0xf991	0xf991
54	0x0000	0xf991	0xf991
55	0x0000	0xf991	0xf991
56	0xcfa8	0xf991	0x3639
57	0xe6b3	0x3639	0xd08a
58	0x7fff	0xd08a	0xaf75
59	0x0000	0xaf75	0xaf75
60	0xcfa8	0xaf75	0x60dd
61	0xe6b3	0x60dd	0x866e
62	0x7fff	0x866e	0xf991
63	0x0000	0xf991	0xf991
64	0x0907	0xf991	0xf096

C.2 Message Partition CRC

The partition message CRC is calculated as follows (assume that the data to be calculated in binary mode is A):

Step 1: $uiCheckSum = payloadLen \wedge 0xEFCDA89$

Step 2: return $uiCheckSum$ as the checksum.

Calculated CRC is 0XEFCDA89.

The following is the intermediate data calculated at each step:

step	payloadLen	uiCheckSum
Input	0x100	0xEFCDA89
Calculation (step1)	0x100	0XEFCDA89
Result	NA	0XEFCDA89

C.3 UIPC message CRC

The black box CRC is calculated as follows (assume that the data to be calculated in binary mode is A):

Note: When calculating the checksum, the RES field and the CRC field are all 0.

Step 1: Pre-set a 16-bit value as the initial value which is 0. The value is marked as an $uiC0$.

Step 2: Take the value of the lowest 1 byte of A, add it to $uiC0$, assign the result to $uiC0$, and remove the lowest 1 byte of A

Step 3: Repeat step 2 until the values of A are all moved.

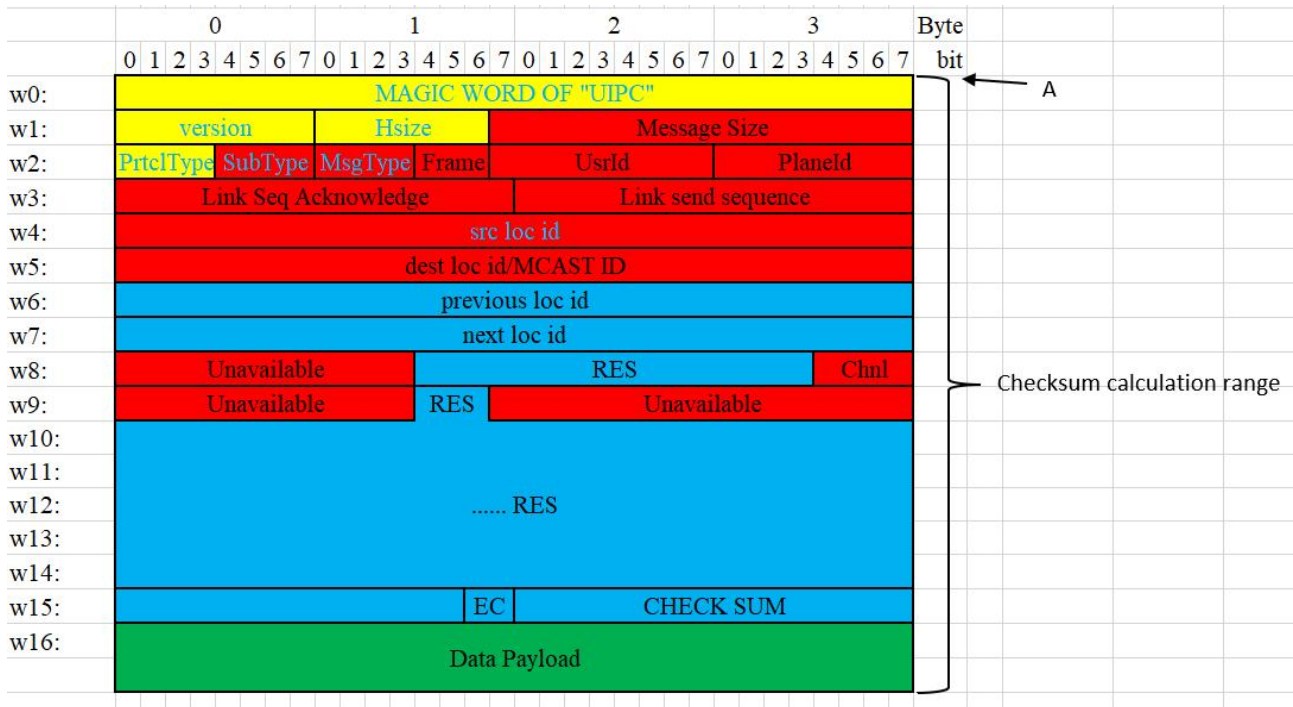
Step 4: Add the high 16bit of $uiC0$ and low 16bit of $uiC0$ and assign the result to $uiC0$.

Step 5: Shift $uiC0$ right by 16bit, add it to $uiC0$, and assign the result to $uiC0$

Step 6: Take the lower 16bit value of uiC0 and assign it to usChecksum.

Step 7: The value of usChecksum is reversed by bit and assigned to usChecksum. The value of the usChecksum is the CRC.

The following is the calculation range of the UIPC message CRC:



The following is the detailed calculation process as example:

pre-condition of this example

The first address of the message: 0x1665b9c

system configuration: host-order, big-endian

Red: the range of the CRC calculation

Blue: calculated CRC value

0x1665b9c:	0x43504955	0x44008005	0x00000001	0x01000000
0x1665bac:	0xffffffff	0x01000000	0xffffffff	0x00000000
0x1665bbc:	0x00000000	0x00000000	0x00000000	0x00000000
0x1665bcc:	0x00000000	0x00000000	0x00000000	0x01f60100
0x1665bdc:	0x00000008	0x00000000	0x00000000	0x00000000

Data value of each element in UIPC message:

Field	network sequence	host sequence	Name	Value	Remark
WORD -0	0x43504955	0x55495043	MAGIC WORD OF "UIPC"	0x55495043	
WORD	0x44008005	0x05800044	version	0x05	24~31 bit

Field	network sequence	host sequence	Name	Value	Remark
-1			Hsize	0x40 (0x80 >> 1)	17~23bit
			Message size	0x44	0~16bit
WORD -2	0x00000001	0x01000000	PrtclType	0x00	MSG_TYPE_ DATA 28~31bit
			MsgPrio	0x01	MSG_PRO_ MEDIUM 24~27bit
			MsgType	0x00	UNICAST 20~23bit
			Frame	0x00	FRM_SINGL E 17~19bit
			UsrId	0x00	8~16bit
			PLaneld	0x00	0~7bit
WORD -3	0x01000000	0x00000001	Link Seq Acknowledge	0x00	16~31bit
			Link send sequence	0x01	0~15bit
WORD -4	0xffffffff	0xffffffff	src loc id	0xffff	
WORD -5	0x01000000	0x00000001	dest loc id	0x00000001	
WORD -6	0xffffffff	0xffffffff	previous loc id	0xffffffff	
WORD -7	0x00000000	0x00000000	next loc id	0x00000000	
WORD -8	0x00000000	0x00000000	Unavailable	0x000	20~31bit
			RES	0x0000	4~19bit
			Chnl	0x0	0~3bit
WORD -9	0x00000000	0x00000000	Unavailable	0	20~31bit
			RES	0	17~19bit
			Unavailable	0	0~16bit
WORD -10~ WORD -14	--	--	RES	--	
WORD -15	0x01f60100	0x0001f601	EC	0x1	16~17bit
			CHECK SUM*	0xf601	0~15bit
AD					

Field	network sequence	host sequence	Name	Value	Remark
WORD-16	0x00000008	0x08000000	Data Payload	0x08000000	

*Note: for calculation of CRC, set CHECK SUM as 0x0000.

Fixed magic number (constant value): 0x0000

A = address of WORD-0

Calculated CRC: 0xf601

Calculated result for each step:

Step 1: Pre-set a 16-bit value as the initial value which is 0. The value is marked as an uiC0.

```
uiC0 = 0x0000
```

Step 2: Take the value of the lowest 1 byte of A, add it to uiC0, assign the result to uiC0, and remove the lowest 1 byte of A

```
uiC0 = uiC0 + (byte)(*A)
```

```
*A = (*A)>>8
```

Step 3: Repeat step 2 until the values of A are all moved.

```
uiC0 = 0x09fe
```

Step 4: Add the high 16bit of uiC0 (0x0000) and low 16bit of uiC0 (0x09fe), and assign the result to uiC0 (0x000009fe)

```
uiC0 = (uiC0 >> 16) + (uiC0 & 0xffff)
```

```
uiC0 = 0x000009fe
```

Step 5: Shift uiC0right by 16bit (0x00000000), add it to uiC0 (0x000009fe), and assign the result (0x000009fe) to uiC0

```
uiC0 += (uiC0 >>16)
```

```
uiC0 = 0x000009fe
```

Step 6: Take the lower 16bit value of uiC0 (0x09fe) and assign it to usCheckSum.

```
usCheckSum = 0x09fe
```

Step 7: The value of usCheckSum is reversed by bit (0xf601) and assigned to usCheckSum. The value of the usCheckSum is the CRC

```
usCheckSum = ~usCheckSum
```

```
usCheckSum = 0xf601
```

Calculated CRC: 0xf601

The following is the intermediate data calculated at each step:

step	loop	*A	uiCheckSum = uiCheckSum + (byte)(*A)
------	------	----	--------------------------------------

step	loop	*A	uiChecksum = uiChecksum + (byte)(*A)
Input	NA	NA	0x00000000
Calculation (step2 - step3)	0	0x55	0x00000055
	1	0x49	0x0000009e
	2	0x50	0x000000ee
	3	0x43	0x00000131
	4	0x05	0x00000136
	5	0x80	0x000001b6
	6	0x00	0x000001b6
	7	0x44	0x000001fa
	8	0x01	0x000001fb
	9	0x00	0x000001fb
	10	0x00	0x000001fb
	11	0x00	0x000001fb
	12	0x00	0x000001fb
	13	0x00	0x000001fb
	14	0x00	0x000001fb
	15	0x01	0x000001fc
	16	0xff	0x000002fb
	17	0xff	0x000003fa
	18	0xff	0x000004f9
	19	0xff	0x000005f8
	20	0x00	0x000005f8
	21	0x00	0x000005f8
	22	0x00	0x000005f8
	23	0x01	0x000005f9
	24	0xff	0x000006f8
	25	0xff	0x000007f7
	26	0xff	0x000008f6
	27	0xff	0x000009f5
	28	0x00	0x000009f5
	29	0x00	0x000009f5
	30	0x00	0x000009f5
	31	0x00	0x000009f5
	32	0x00	0x000009f5
	33	0x00	0x000009f5
	34	0x00	0x000009f5
	35	0x00	0x000009f5
	36	0x00	0x000009f5
	37	0x00	0x000009f5
	38	0x00	0x000009f5
39	0x00	0x000009f5	

step	loop	*A	uiChecksum = uiChecksum + (byte)(*A)
	40	0x00	0x000009f5
	41	0x00	0x000009f5
	42	0x00	0x000009f5
	43	0x00	0x000009f5
	44	0x00	0x000009f5
	45	0x00	0x000009f5
	46	0x00	0x000009f5
	47	0x00	0x000009f5
	48	0x00	0x000009f5
	49	0x00	0x000009f5
	50	0x00	0x000009f5
	51	0x00	0x000009f5
	52	0x00	0x000009f5
	53	0x00	0x000009f5
	54	0x00	0x000009f5
	55	0x00	0x000009f5
	56	0x00	0x000009f5
	57	0x00	0x000009f5
	58	0x00	0x000009f5
	59	0x00	0x000009f5
	60	0x00	0x000009f5
	61	0x01	0x000009f6
	62	0x00	0x000009f6
	63	0x00	0x000009f6
	64	0x08	0x000009fe
	65	0x00	0x000009fe
	66	0x00	0x000009fe
	67	0x00	0x000009fe
uiC0 = (uiC0 >> 16) + (uiC0 & 0xffff)	NA	NA	0x000009fe
uiC0 += (uiC0 >> 16)	NA	NA	0x000009fe
usChecksum = (VOS_UINT16) (~ uiC0)	NA	NA	0xf601

C.4 Black box CRC

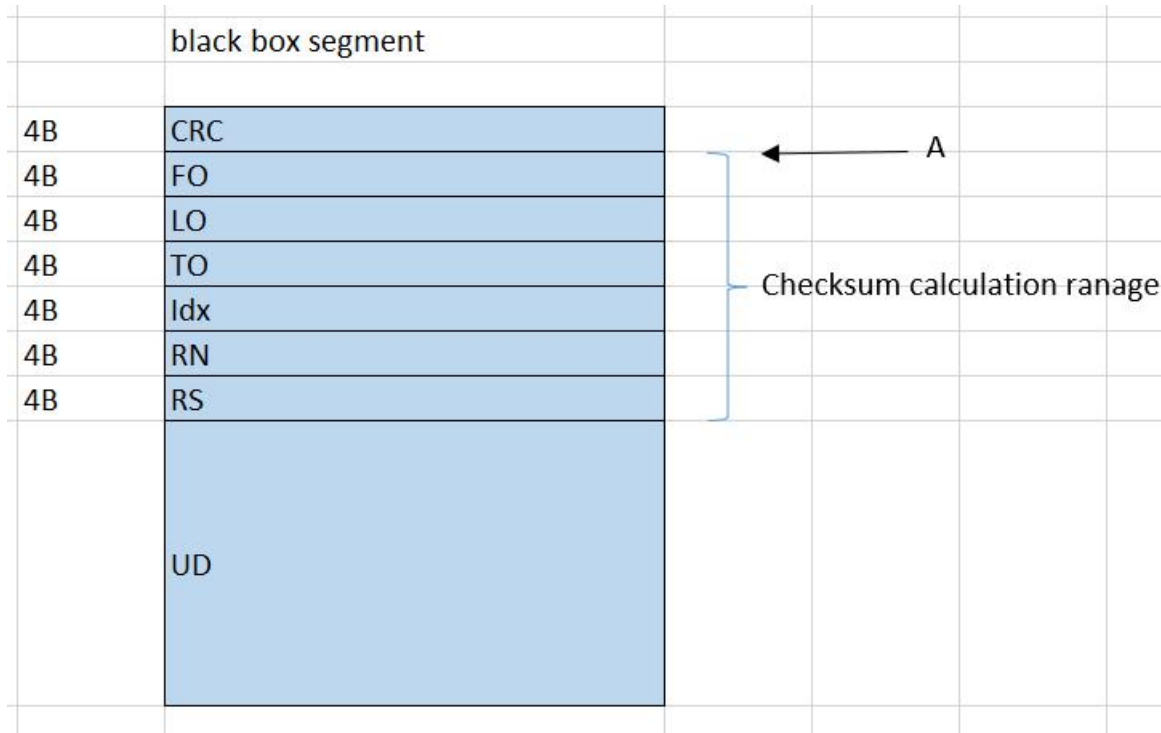
The black box CRC is calculated as follows (assume that the data to be calculated in binary mode is A):

Step 1: Pre-set a 32-bit value as the initial value which is 0. The value is marked as the uiChecksum.

Step 2: Add the lower four bytes of A to uiCheckSum, save the result to the uiCheckSum, and shift A 32 bits to the right and save the result to A.

Step 3: Repeat step 2 until values of A are all removed. The value of the uiCheckSum is the CRC value of the black box record.

The following is the calculation range of the black box segment CRC:



The following is the detailed calculation process:

The first address of the segment: 0xd79d8f44

Red: the range of the CRC calculation

Blue: CRC calculated value

host-order, little-endian

```

0xd79d8f44:  0x00020248  0x0000001c  0x00000114  0x00000114
0xd79d8f54:  0x00000002  0x00000002  0x00020000  0x4f14b13b
0xd79d8f64:  0x00000000  0x000000da  0x00000001  0x21130002
0xd79d8f74:  0x00000000  0x00000114  0x00000000  0x21130002
0xd79d8f84:  0x000000a6  0x00000000  0x00000000  0x00000000
0xd79d8f94:  0x00000000  0x00000000  0x00000000  0x0ceeaea4
0xd79d8fa4:  0xffffffff  0xffffffff  0x00000000  0x21130002
0xd79d8fb4:  0x00000000  0x00000000  0x00000000  0x00020401
0xd79d8fc4:  0x21130002  0x635f736f  0x74707570  0x2e6b7361
0xd79d8fd4:  0x00000063  0x00000000  0x00000000  0x00000000
0xd79d8fe4:  0x00000000  0x000a00a7  0x00000000  0x00000000
0xd79d8ff4:  0x00000000  0x0969c2d7  0x09875c85  0x09876501
0xd79d9004:  0x096bd262  0x0965979e  0x09656f9b  0x0965711f
0xd79d9014:  0x096a3c58  0x0966e757  0x09669887  0x504f445b
    
```

0xd79d9024: 0x762d4152 0x7043736f 0x65477075 0x73615474
 0xd79d9034: 0x746f546b 0x69546c61 0x79426b63 0x61747348

Data value of each element in black box segment:

Title	Parse	Value
CRC	CRC check code	0x00020248
FO	Offset of the first record	0x0000001c
LO	Offset of the last record	0x00000114
TO	Offset of the tail record	0x00000114
Idx	Serial number	0x00000002
RN	Number of records written in the segment	0x00000002
RS	Size of the segment	0x00020000

Fixed magic number (constant value): 0x00000000

A = address of FO

Calculated CRC: 0x00020248

Calculated result for each step:

Step 1: Pre-set a 32-bit value as the initial value which is 0. The value is marked as the uiChecksum.

uiChecksum = 0x00000000

Step 2: Add the lower four bytes at address A (0x0000001c) to uiChecksum (0x0000001c), save the result to the uiChecksum (0x0000001c)

uiChecksum = uiChecksum + *A

Add 4 to address A, to point A to the next address (0x00000114).

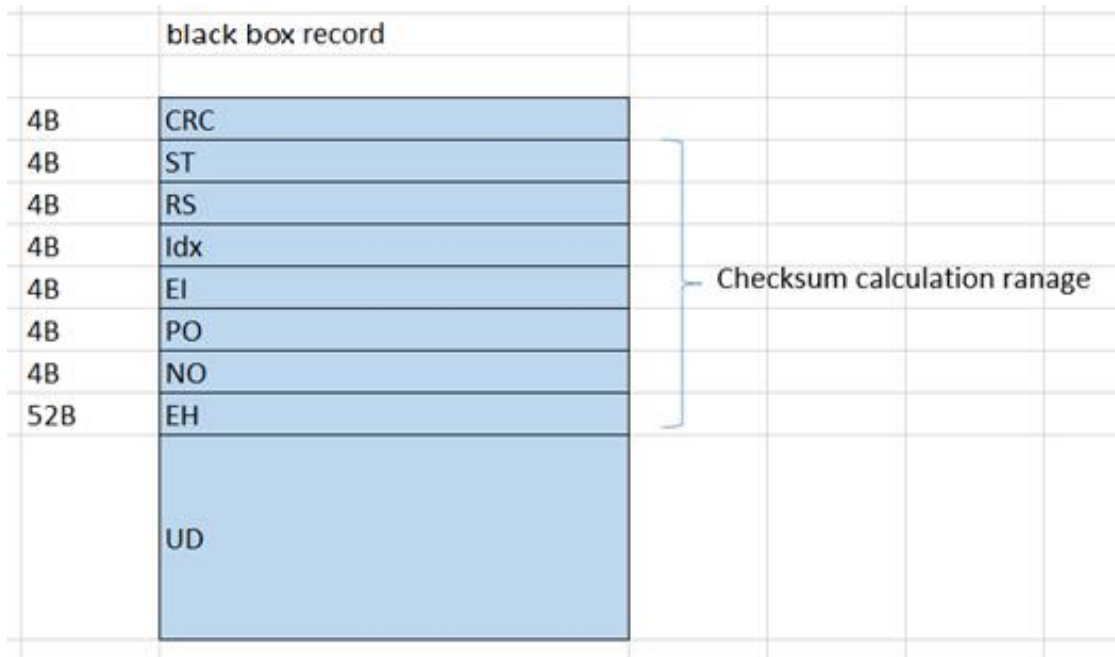
Step 3: Repeat step 2 until values of A are all removed. The value of the uiChecksum is the CRC value of the black box segment.

uiChecksum = 0x00020248

The following is the intermediate data calculated at each step:

step	loop	*A	uiChecksum
Input	NA	0x0000001c	0x00000000
Calculation (step2 - 3)	0	0x0000001c	0x0000001c
	1	0x00000114	0x00000130
	2	0x00000114	0x00000244
	3	0x00000002	0x00000246
	4	0x00000002	0x00000248
	5	0x00020000	0x00020248
CRC	NA	NA	0x00020248

The CRC calculation process of the black box record of audit log is consistent with the black box segment, and the calculation range is as follows:



The following is the detailed calculation process as example:

The first address of black box record: 0xd78a1cb8

Red: the range of the CRC calculation

Blue: CRC calculated value

host-order, little-endian

```

0xd78a1cb8: 0x22fde069 0x00000000 0x00000040 0x00000001
0xd78a1cc8: 0x00000300 0x00000000 0x00000000 0x00000000
0xd78a1cd8: 0x00000300 0x0000000c 0x00000000 0x00000000
0xd78a1ce8: 0x100407e6 0x060b230e 0x00000084 0x00000002
0xd78a1cf8: 0x0ceaea4 0xffffffff 0xffffffff 0x00000000
0xd78a1d08: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d18: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d28: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d38: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d48: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d58: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d68: 0x00000000 0x00000000 0x00000000 0x00000000
0xd78a1d78: 0x00000000 0x00000000 0x00000000 0x00000000
    
```

Data value of each element in black box record:

Title	Parse	Value
CRC	CRC check code	0x22fde069
ST	Number of system startups	0x00000000

Title	Parse	Value
RS	Size of the record	0x00000040
Idx	Serial number	0x00000001
EI	Event id	0x00000300
PO	Offset from the previous record	0x00000000
NO	Offset to the next record	0x00000000
EH	Black box event public header information	0x00000000 0x00000300 0x0000000c 0x00000000 0x00000000 0x100407e6 0x060b230e 0x00000084 0x00000002 0x0ceeaea4 0xffffffff 0xffffffff 0x00000000

Fixed magic number (constant value): 0x00000000

A = address of ST

Calculated CRC: 0x22fde069

Calculated result for each step:

Step 1: Pre-set a 32-bit value as the initial value which is 0. The value is marked as the uiChecksum.

uiChecksum = 0x00000000

Step 2: Add the lower four bytes at address A (0x00000000) to uiChecksum (0x00000000), save the result to the uiChecksum (0x00000000)

uiChecksum = uiChecksum + *A

Add 4 to address A, to point A to the next address (0x00000040).

Step 3: Repeat step 2 until values of A are all removed. The value of the uiChecksum is the CRC value of the black box segment.

uiChecksum = 0x22fde069

The following is the intermediate data calculated at each step:

step	loop	*A	uiChecksum
Input	NA	0x00000000	0x00000000
Calculation (step2 - 3)	0	0x00000000	0x00000000
	1	0x00000040	0x00000040
	2	0x00000001	0x00000041
	3	0x00000300	0x00000341
	4	0x00000000	0x00000341
	5	0x00000000	0x00000341
	6	0x00000000	0x00000341
	7	0x00000300	0x00000641
	8	0x0000000c	0x0000064d
	9	0x00000000	0x0000064d
10	0x00000000	0x0000064d	

step	loop	*A	uiCheckSum
	11	0x100407e6	0x10040E33
	12	0x060b230e	0x160F3141
	13	0x00000084	0x160F31C5
	14	0x00000002	0x160F31C7
	15	0x0ceeaea4	0x22FDE06B
	16	0xffffffff	0x22FDE06A
	17	0xffffffff	0x22FDE069
	18	0x00000000	0x22FDE069
CRC	NA	NA	0x22fde069