Application Note

Document No.: AN1139

G32R501 Secure Boot Application Note

Version: V1.1

1 Introduction

This application note introduces G32R501 secure boot, including design principles for secure-boot applications and how to use the secure-boot examples in the G32R5xx SDK.

Contents

2 Overview of G32R5xx Secure Boot

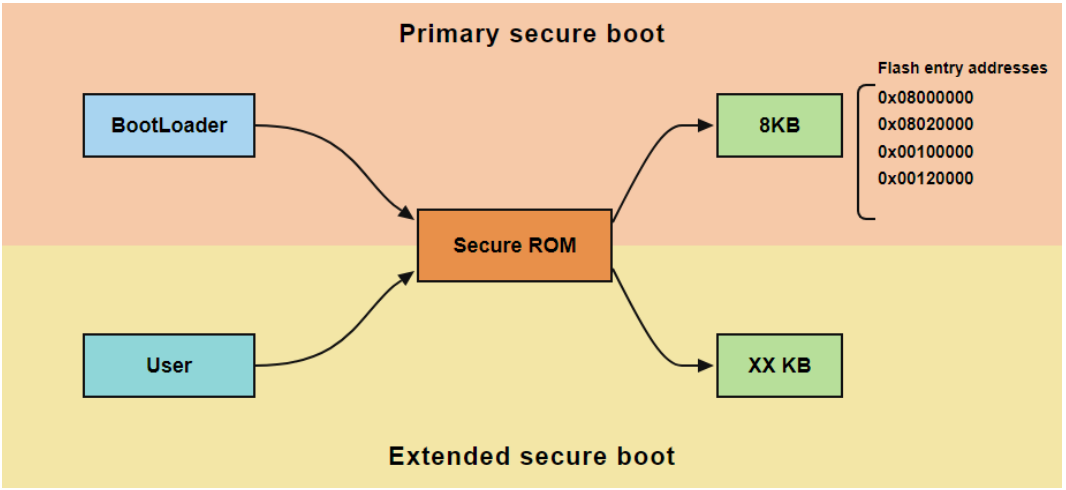
The G32R5xx secure-boot feature is implemented by on-chip BootROM code. It uses a 128-bit AES-CMAC authentication algorithm on the first 8 KB of user firmware in Flash. Only after authentication passes does the device jump to the user application.

Authentication of the first 8 KB by BootROM is called “primary secure boot”. Authentication beyond that 8 KB range is called “extended secure boot” and is optionally started by user application code.

The Flash sector checked by primary secure boot must be set to Zone1-ExeOnly. The Flash sector checked by extended secure boot may be Unsecure, Zone1-Secure, or Zone1-ExeOnly. In other words, Flash sectors used for CMAC authentication must not be configured with Zone2 permissions.

The primary and extended secure-boot check ranges are as shown in Figure 1.

Figure 1 Primary and extended secure-boot check diagram



3 Secure Boot Option

G32R5xx secure boot supports boot from CPU0 only. Table 1 lists all secure-boot entry addresses, the corresponding BootMode values, and where the Golden CMAC Tag must be stored for each entry.

Table 1 Secure boot option configuration

Option	BootMode value	Flash entry point	Flash block/sector	Golden CMAC Tag storage location
0	0x0A	0x08000000	Bank 0, Sector 0 (Busmatrix IF)	0x08000008
1	0x2A	0x00100000	Bank 0, Sector 0 (ITCM IF)	0x00100008
2	0x4A	0x08020000	Bank 1, Sector 0 (Busmatrix IF, DualBank Mode)	0x08020008
3	0x6A	0x00120000	Bank 1, Sector 0 (ITCM IF, DualBank Mode)	0x00120008

Note: The secure-boot entry sector must be configured as Zone1-ExeOnly.

4 How to Design Secure Boot Applications

Applications that use secure boot must meet the requirements in this chapter so that authentication is correct and the application runs properly.

4.1 Golden CMAC Tag Storage Location

The Golden CMAC Tag is the expected 16-byte Tag stored in firmware for CMAC authentication. For primary secure boot, the Tag start address must be 8 bytes after the secure-boot entry address. For example, if the entry is 0x08000000, the Golden CMAC Tag starts at 0x08000008.

For extended secure boot with a user-defined Flash check range, the Tag start address is not fixed, but must meet:

- The extended-boot Golden CMAC Tag must not lie inside the primary secure-boot check range.
- The Tag start address must be 4-byte aligned.
- The Tag storage area must lie inside the CMAC check range. For example, if the entire Flash is checked, the Tag must be stored in Flash.

4.2 Requirements for Encrypted Sector Settings of Applications to be Checked

When a Flash sector that holds CMAC-authenticated code is set to an encrypted permission, the CPU may execute from that region but must not read its data. Therefore, any Flash data the CPU must read must not be placed in an encrypted sector.

Typical Flash data the CPU must read includes read-only data (for example “const” globals) and initial values of non-zero-initialized globals (copied from Flash to RAM at start). Code regions that must not be encrypted include:

- CONST (RO-DATA): read-only data.
- RW: read/write data. Non-zero-initialized globals are stored with other data in Flash and copied to RAM after power-up.

Therefore, the only Flash segment that may be encrypted is the text (code) segment. Flash sectors that hold other data segments must not be encrypted.

4.3 Application Design Example (MDK-ARM)

Based on Sections 4.1 and 4.2, this section uses MDK-ARM as an example of designing an application that meets secure-boot requirements.

4.3.1 Design example of Golden CMAC Tag value storage location

Section 4.1 described Tag location requirements; this subsection shows how to implement them in code.

1. Primary secure boot Golden CMAC Tag storage design:

Per the Arm Cortex-M architecture, the first 8 bytes at the start address must hold SP and PC. Therefore the Golden CMAC Tag is stored immediately after SP/PC—8 bytes past the entry address. In addition, the primary secure-boot entry sector must be Zone1-ExeOnly. To keep the interrupt vector table unencrypted, the remaining vectors must not live in the entry sector. For these reasons, two interrupt vector tables are used. The first holds SP, PC, and the Golden CMAC Tag; the second holds all other interrupt vectors. Example of the vector table that stores the Golden CMAC Tag:

```
NO_OPTIMIZE_VAR const PINT __VECTOR_TABLE_CMAC[6] __VECTOR_TABLE_ATTRIBUTE = {
    [0]   = (PINT)(&__INITIAL_SP),
    [1]   = Reset_Handler,
    [2]   = (PINT)0xFFFFFFFF,
    [3]   = (PINT)0xFFFFFFFF,
    [4]   = (PINT)0xFFFFFFFF,
    [5]   = (PINT)0xFFFFFFFF,
};
```

This vector table must be placed at the boot address. The other vector table holds all interrupt vectors. Its location is flexible, but its sector must not be encrypted. Example (excerpt):

```
const PINT __VECTOR_TABLE[512] = {
    [0]   = (PINT)(&__INITIAL_SP),
    [1]   = Reset_Handler,
    [2]   = NMI_Handler,
    [3]   = HardFault_Handler,
    /* ... remaining vectors see SDK example ... */
};
```

2. Extended secure boot Golden CMAC Tag storage design:

Users may authenticate data beyond 8 KB; user-initiated authentication is extended secure boot. Example matching the Tag location rules above:

```
# if defined (__ARMCC_VERSION) || defined (__CC_ARM) || defined (__ICCARM__)
    __attribute__((__used__, section(".ARM.__at_0x08008000")))
```

```
# elif defined (__GNUC__) && !defined (__ARMCC_VERSION)
    __attribute__((section(".cmac_all")))

# endif
const uint32_t cmac_all[] =
{
    0xFFFFFFFF,
    0xFFFFFFFF,
    0xFFFFFFFF,
    0xFFFFFFFF,
};
```

This array reserves a fixed location in the compiled bin so the compiler does not reuse the Tag storage for other data. Address 0x08008000 is the Golden CMAC Tag start address in the example.

4.3.2 Link file design example

As described in Section 4.2, only the code segment in the compiled bin may be encrypted. Secure-boot applications therefore need a redesigned linker file that gathers non-encryptable segments into fixed Flash sectors.

Note: Do not set encryption on Flash sectors that hold non-encryptable segments.

In the sct example for this section, LR_SECURE_ROM holds only the code segment and the first vector-table array; that Flash sector may be encrypted. LR_UNSECURE_ROM holds read-only data, “__main”, RW data, and related content; that sector must not be encrypted, or the program will not run correctly.

MDK-ARM sct link-file design example (excerpt):

```
LR_SECURE_ROM __RO_BASE __RO_SIZE {
  ER_SECURE_ROM __RO_BASE __RO_SIZE {
   *.o (RESET, +First)
   .ANY (+RO)
   .ANY (+XO)
  }
}
LR_UNSECURE_ROM __CPU0_UNSECURE_ROM_BASE __CPU0_UNSECURE_ROM_SIZE {
  ER_UNSECURE_ROM __CPU0_UNSECURE_ROM_BASE __CPU0_UNSECURE_ROM_SIZE {
    *(InRoot$$Sections)
    .ANY (+CONST)
  }
```

```
  /* ... RW/ZI segments see SDK example sct ... */
}
```

See the sct file of the corresponding secure-boot example in the G32R5xx SDK for the full content.

5 How to Use G32R5xx Secure Boot Function

5.1 Operation Process of Secure Boot

Besides following Chapter 4 design rules, use the following process:

1. Configure the secure-boot start sector as Zone1 ExeOnly.
2. Configure the chip CMACKEY. CMACKEY is stored in OTP at 0x0810B020, length 128 bytes. Factory default is all FF.
3. From the compiled raw application bin and the user CMACKEY, use Geehy_Bin to generate a bin that contains the Golden CMAC Tag.
4. Program/download the bin with the Golden CMAC Tag to the chip.
5. Per Table 1, set the required BootMode (during debug, prefer simulated BootMode for secure boot).
6. Press reset or power-on reset to run the application.

5.2 Geehy_Bin Tool

After compiling the raw application bin, use Geehy_Bin to generate a bin with the Golden CMAC Tag for secure boot.

This section only covers generating a bin with the Golden CMAC Tag. For more tool usage, see AN1131 *G32R5xx Tool User Manual*.

5.2.1 key.txt and cmd files

Geehy_Bin needs a text file with the CMACKEY and length information for primary or extended secure boot. The G32R5xx SDK provides examples under utilities/geehy_tool: key.txt, test_major.cmd, and test_extend.cmd.

- key.txt

Specifies the user CMACKEY. The sample content is the chip default CMACKEY.

```
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Note: If you change the chip CMACKEY, this file must match the value programmed inside the chip.

- test_major.cmd

Specifies the primary secure-boot entry address and check length. Primary check length is fixed at 0x2000. Example content:

```
ROMS
{
```

```
  FLASH: o=0x08000000 l=0x2000, fill = 0xFFFF
}
```

Here o=0x08000000 is the primary check entry and must match the BootMode entry; l=0x2000 is the fixed primary check length.

- test_extend.cmd

Specifies the user-defined check start address and length. Example content:

```
ROMS
{
  FLASH: o=0x08000000 l=0x10000, fill = 0xFFFF
}
```

Here, “o=0x08000000” specifies the start address of the extended secure boot check (that is, the user-defined authentication), and “l=0x10000” specifies the check length. The start address and length configured for extended secure boot must match those set in the application. In addition, the extended secure boot check range may include the primary secure boot check range of “0x2000”.

5.2.2 Command Examples

- Generate a primary secure-boot bin

```
Geehy_Bin.exe test_major.cmd -m --load_image --cmac=key.txt project.bin -tag=0x08000008 -o
project_cmac_major.bin
```

“-tag=0x08000008” sets where the computed Golden CMAC Tag is stored. For primary secure boot this is always entry+8 bytes.

- Generate an extended secure-boot bin

```
Geehy_Bin.exe test_extend.cmd -e --load_image --cmac=key.txt project_cmac_major.bin -
tag=0x08008000 -load=0x08000000 -o project_cmac_extend.bin
```

For extended secure boot, the “-tag” address must match the application Tag storage address and must not lie in the primary check range.

Note: Extended secure boot is optional. If both primary and extended secure boot are used, generate the primary bin first, then use that bin to generate the combined primary and extended bin.

5.3 G32R5xx SDK Secure Boot Example

The G32R5xx SDK provides a secure-boot sample that covers primary 8 KB authentication and

how to authenticate code beyond 8 KB.

Path: driverlib\g32r501\examples\eval\boot\boot_ex3_cpu0_secure_flash.

The example assumes Flash encryption permissions are already set correctly (entry sector Zone1 EXEONLY) and uses the default CMACKEY. For Flash sector permissions and CMACKEY updates, see DCS module examples and documentation.

The following uses MDK-ARM to run the example.

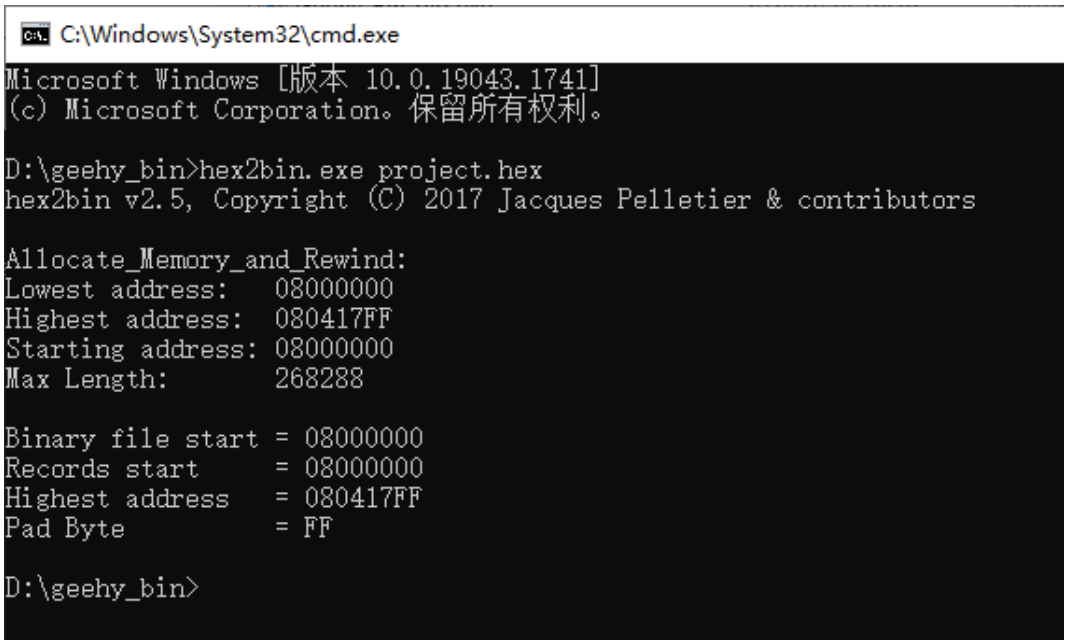
5.3.1 Generate the bin file with Golden CMAC Tag value

Geehy provides tools to generate a bin with the Golden CMAC Tag.

1. Generate a bin without Golden CMAC Tag from a hex file

Secure-boot applications use an sct with multiple load regions. The built-in MDK bin generator cannot produce one complete bin (it emits split bins). Use hex2bin instead. The tool is at SDK “utilities/geehy_tool/hex2bin.exe”; run it in the output directory that contains the target “.hex”, or pass absolute paths for input/output. See Figure 2.

Figure 2 Generate a bin without Golden CMAC Tag



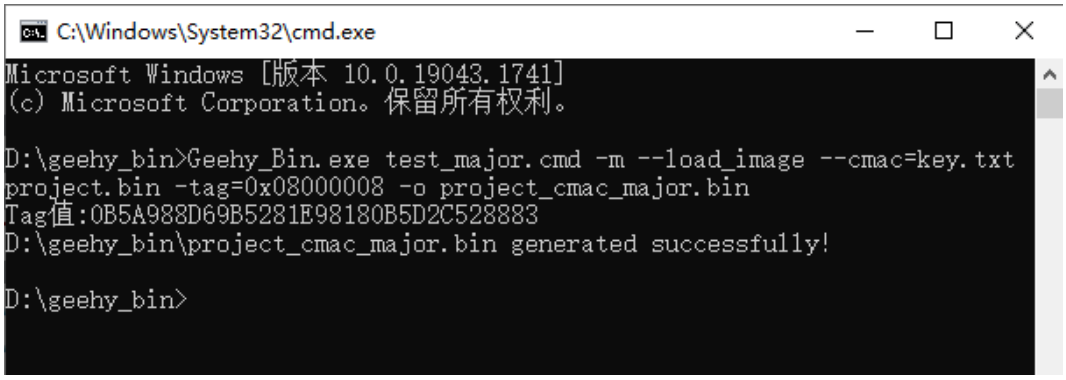
2. Use Geehy tools to generate a bin with Golden CMAC Tag

First generate the primary secure-boot Tag bin. Enter SDK “utilities/geehy_tool/” (contains “key.txt”, “test_major.cmd”, “test_extend.cmd”, and “Geehy_Bin.exe”), place the previous “project.bin” in the working directory or use a relative path, then run:

```
Geehy_Bin.exe test_major.cmd -m --load_image --cmac=key.txt project.bin -tag=0x08000008 -o
```

```
project_cmac_major.bin
```

Figure 3 Generate a bin with Golden CMAC Tag



Then use that primary Tag bin to generate the extended Tag bin:

```
Geehy_Bin.exe test_extend.cmd -e --load_image --cmac=key.txt project_cmac_major.bin -tag=0x08008000 -load=0x08000000 -o project_cmac_extend.bin
```

Figure 4 Generate extended secure-boot Golden CMAC Tag bin

Always generate the primary Tag bin first, then the extended Tag bin from it.

Note: Tools and files used in this section are available in the G32R5xx SDK.

5.3.2 Change the boot mode to secure boot

Factory default boot is not secure boot; configure secure-boot mode yourself. Boot mode can be set by programming OTP or by simulated boot. OTP can be written only once; during debug, prefer simulated boot. The following uses simulated boot to select secure boot.

1. Connect a J-Link Debugger to the board.
2. Use the J-Link Commander command-line tool to connect to the MCU. After a successful connection, the display appears as shown in Figure 5.

Figure 5 J-Link Commander successfully connected to G32R501

```
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: CORTEX-M52
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "CORTEX-M52" selected.


Connecting to target via SWD
Found SW-DP with ID 0x6BA02477
DPIDR: 0x6BA02477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[4]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x84770001)
AP[1]: AHB-AP (IDR: 0x84770001)
AP[2]: AHB-AP (IDR: 0x84770001)
AP[3]: APB-AP (IDR: 0x54770002)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x630FD241. Implementer code: 0x63 (ARM China)
Feature set: Mainline
Cache: L1 I/D-cache present
Found Cortex-M52 r0p1, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
Security extension: not implemented
CoreSight components:
ROMTbl[0] @ E00FF000
[0][0]: E000E000 CID B105900D PID 000F5D24 DEVARCH 47702A04 DEVTYPE 00 ???
[0][1]: E0001000 CID B105900D PID 000F5D24 DEVARCH 47711A02 DEVTYPE 00 DWT
[0][2]: E0002000 CID B105900D PID 000F5D24 DEVARCH 47701A03 DEVTYPE 00 FPB
[0][3]: E0000000 CID B105900D PID 000F5D24 DEVARCH 47701A01 DEVTYPE 43 ITM
[0][5]: E0041000 CID B105900D PID 000F5D24 DEVARCH 47754A13 DEVTYPE 13 ETM
[0][6]: E0003000 CID B105900D PID 000F5D24 DEVARCH 47700A06 DEVTYPE 16 ???
[0][7]: E0042000 CID B105900D PID 000F5D24 DEVARCH 47701A14 DEVTYPE 14 CSS600-CTI
[0][8]: E0046000 CID B105900D PID 001BB9BA DEVARCH 47710A55 DEVTYPE 55 ???
I-Cache L1: 4 KB, 64 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 4 KB, 32 Sets, 32 Bytes/Line, 4-Way
Memory zones:
  Zone: "Default" Description: Default access mode
Cortex-M52 identified.
J-Link>
J-Link>
```

Note: J-Link Commander must support Cortex-M52. This demonstration uses V7.96h. The J-Link probe must also support Cortex-M52 debug.

1. Configure secure boot with entry 0x08000000 (BootMode 0x0A). In J-Link Commander enter:

```
w4 0x50020000 0x5AFFFFFF
w4 0x50020004 0xFFFFFF0A
w4 0x50020008 0xFFFFFFFF
```

5.3.3 Download and run routines

Use J-Flash to download the bin that contains both primary and extended Golden CMAC Tags.

Note: G32R5xx has DCS encryption protection; decrypt before programming Flash. See DCS documentation for decryption.

After download, press reset (do not power off, or the simulated boot mode is lost) and observe the board LEDs:

- LED2 and LED3 off: secure boot failed.
- LED2 or LED3 blinking: secure boot passed; user extended CMAC check failed.
- LED2 and LED3 blinking: secure boot passed and user extended CMAC check passed.

6 Revision

Table 2 Document revision

Date	Version	Change history
2025.1	1.0	● Initial release
2026.8	1.1	● Applicable products listed as the series with Secure Boot capability (G32R501).

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “TM” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy

hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY

DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS

"AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.