MT793X IoT SDK for Secure Boot How-To

Version: 1.1

Release date: 2022-07-11

Version History

Version	Date	Author	Description
1.0	2021-08-11	Anthony Liu	Initial release
1.1	2022-07-11	CY Chan	Modify for new EDCSA p384/p521 algorithm

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 A Real Example of Secure Boot Chain

This is a quick example for those that cannot wait to see how secure boot works.

```
loader disable 30s timeout
loader init
 0
Your choose c
scott_image_init: 33cc!
sboot: rtos 7 addr 0x18044080 size 0x2fff80
jump pc 0x180c1ef5, sp 0x111000
hal_psram_init

Psram type : 2
 ASIC_MPU disable psram protection in callback
```

The message lines starting with ‘sboot’ and ‘scott’ are from secure boot.

1.2 Scope

This document describes the usage of Chain-of-Trust of the MT7931/MT7933.

1.2.1 Chain-of-Trust

The Chain-of-Trust is a security design pattern that can be found everywhere in embedded systems with security in mind. It works by planting security information in the hardware that cannot be changed after its implantation and the firmware can validate another firmware later before using it. The design is called ‘chain’ because the procedure can be cascaded several times and the trust of program can be propagated along the cascade. Figure 1 shows the overview of this procedure in the MT793X.

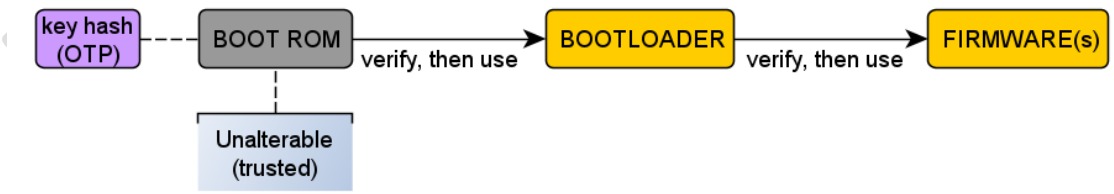


Figure 1: MT793X Chain-of-Trust

2 SDK Default Setting

Since SDK release 1.0, BOOTLOADER, TF-M, and application firmware images are signed with ECC by default.

2.1 The Default Key

For those that are curious, the default key pair of EDCSA p256 algorithm in SDK release is:

```
$ cat project/mt7933_hdk/apps/bootloader/mtk-dev.pem
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgmyTNAJ4KL3dx15E2
7GdJ7+FeVnY29dXJVsA0dwHNngKhRANCAAQ5SQ5u+SgoOO8qaN2A2171kZYGL+cr
91/SzV7ln5lDWJJEYvn41lADOQb5mEYCzfBt8/q88ObveGoiVqVRTo8s
-----END PRIVATE KEY-----
$ cat project/mt7933_hdk/apps/bga_sdk_demo/mtk-dev.pem
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgmyTNAJ4KL3dx15E2
7GdJ7+FeVnY29dXJVsA0dwHNngKhRANCAAQ5SQ5u+SgoOO8qaN2A2171kZYGL+cr
91/SzV7ln5lDWJJEYvn41lADOQb5mEYCzfBt8/q88ObveGoiVqVRTo8s
-----END PRIVATE KEY-----
```

The list above shows it contains a private key, rather than a pair of private and public keys. The reason is that in ECC (elliptic curve cryptography), the public key can always be derived from its corresponding private key.

Note there are identical copies of the same key in each project folders to simplify the dependencies between different project folders at build time.

2.2 Signing in Makefiles

During the build, Makefile of BOOTLOADER signs BOOTLOADER with keys be selected (by files exist on locate path) of mtk-dev.pem/mtk-dev384.pem/mtkdev521.pem that corresponding to EDCSA p256/p384/p521 algorithm.

```
FILE: project/mt7933_hdk/apps/bootloader/GCC/Makefile

SGN_PEM    = $(SDK_PATH)/$(APP_PATH)/mtk-dev.pem
SGN_PEM384 = $(SDK_PATH)/$(APP_PATH)/mtk-dev384.pem
SGN_PEM521 = $(SDK_PATH)/$(APP_PATH)/mtk-dev521.pem

<…abbreviated>
        $(IMGTOOL) sign --pad-header --header-size $(HEADER_OFFSET)         \
                        --load-addr $(LOAD_ADDR_RAM) -S 65536               \
                        -k $(SGN_PEM) -k384 $(SGN_PEM384) -k521 $(SGN_PEM521)\
                        --align 4 -v $(SGN_VER) --pubkey                     \
                        --infra-dapc "$(BL_IMAGE_TLV_INFRA_DAPC)"            \
                        --aud-dapc "$(BL_IMAGE_TLV_AUDIO_DAPC)"              \
                        --asic-mpu "$(BL_IMAGE_TLV_ASIC_MPU)"                \
                        --no-bootrom-log -t SIGN_ALL                        \
                        $< $@ >> $(BUILD_LOG) 2>>$(ERR_LOG)
```

And Makefile of application image (such as bga_sdk_demo/qfn_sdk_demo) signs itself with be selected keys too.

```
FILE: project/mt7933_hdk/apps/bga_sdk_demo/GCC/Makefile

SGN_PEM = $(SDK_PATH)/$(APP_PATH)/mtk-dev.pem
SGN_PEM384 = $(SDK_PATH)/$(APP_PATH)/mtk-dev384.pem
SGN_PEM521 = $(SDK_PATH)/$(APP_PATH)/mtk-dev521.pem

<…abbreviated>
        $(IMGTOOL) sign --pad-header --header-size $(HEADER_SGN_SZ)         \
                        --load-addr $(LOAD_ADDR_XIP) -S 3145600             \
                        -k $(SGN_PEM) -k384 $(SGN_PEM384) -k521 $(SGN_PEM521) \
                        --align 4 -v $(SGN_VER) --pubkey -t SIGN_ALL         \
                        $$< $$@ >> $(BUILD_LOG) 2>>$(ERR_LOG)
```

The default build output then have signatures added to the firmware images. For example, in a build of the project folder bga_sdk_demo, the output files can be found like below:

```
$ ./build.sh mt7933_hdk bga_sdk_demo bl

<…abbreviated>

$ ls out/mt7933_hdk/bga_sdk_demo/*.sgn
out/mt7933_hdk/bga_sdk_demo/mt7931an_bootloader-ram.sgn
out/mt7933_hdk/bga_sdk_demo/mt7931an_bootloader-xip.sgn
out/mt7933_hdk/bga_sdk_demo/mt7933cv_xip_bga_al.sgn
```

2.3 Disable Signing

The SDK enables ECC signing by default, but the signing can be turned off if an application does not need it. You can disable signing by tuning feature options:

FILE	**project/mt7933_hdk/apps/bootloader/GCC/feature.mk**	
MTK_SECURE_BOOT_ENABLE	y/n	Set to ‘y’ to enable verification against firmware blobs (TF-M/APP). Set to ‘n’ to disable verification against firmware blobs (TF-M/APP).
MTK_BL_SIGN_ENABLE	y	Set to ‘y’ to enable signing of BOOTLOADER itself. Set to ‘n’ to disable signing of BOOTLOADER itself. (please do not try this) Some settings are provided by the wrapper of BOOTLOADER, and, therefore, always keep this option ‘y’.
MTK_BL_SECURE_BOOT_PERMISSIVE_ENABLE	y/n	If the value is set to ‘y’, when BOOTLOADER fails to verify a firmware blob, you get a warning and the boot up process continues. Set to ‘n’ if strict hang should happen when a verification fails.
FILE	**project/mt7933_hdk/apps/bga_sdk_demo/GCC/feature.mk** **project/mt7933_hdk/apps/qfn_sdk_demo/GCC/feature.mk**	
MTK_RTOS_SIGN_ENABLE	y/n	Set to ‘y’ to enable signing of application firmware (RTOS) itself. Set to ‘n’ to disable signing. If the value is set to ‘y’, make sure MTK_SECURE_BOOT_ENABLE is also ‘y’.
MTK_SECURE_BOOT_ENABLE	y/n	Set to ‘y’ to enable secure boot function. Otherwise, enabling MTK_RTOS_SIGN_ENABLE causes build failure. Set to ‘n’ to disable secure boot function.

3 How to Self Sign

If the default key in the SDK is to be replaced with a self-generated key, follow the steps below:

Table 1. Steps to Self-Signing

1.	Generate an ECC key pair.
2.	Replace the *mtk-dev.pem/ mtk-dev384.pem/ mtk-dev521.pem*.
3.	Rebuild the firmware images with the new *mtk-dev.pem/ mtk-dev384.pem/ mtk-dev521.pem*.
4.	Generate the hash of the public key in *mtk-dev.pem/ mtk-dev384.pem/ mtk-dev521.pem*.
5.	Blow eFuse in the MT793X with the hash.
6.	Download the newly built firmware into flash.

Notes:

1. Keep the generated key pair well. There is no known backdoor to restore a missing key.
2. Keep the private key safe from access by unauthorized individuals.

The steps above are described in following sections.

3.1 Generate ECC Key Pair

Use OpenSSL to generate an ECC key pair for ECDSA p256 algorithm .

```
$ openssl ecparam -genkey -name prime256v1 -out my256.pem
```

This command generates a file, called *my256.pem*, with an ECC key pair (private and public keys) in it.
generate an ECC key pair for ECDSA p384 algorithm:

```
$ openssl ecparam -genkey -name secp384r1 -out my384.pem
```

generate an ECC key pair for ECDSA p521 algorithm

```
$ openssl ecparam -genkey -name secp521r1 -out my521.pem
```

3.2 Replace the *mtk-dev.pem/ mtk-dev384.pem/ mtk-dev521.pem*

Just copy *my256.pem/my384.pem/my521.pem* to the location of *mtk-dev.pem/mtk-dev384.pem/mtk-dev521.pem* files and rename it to replace *mtk-dev.pem/mtk-dev384.pem/mtk-dev521.pem*. Clear the previous build output and build again to create new firmware images signed with the new key.

3.3 Generate the Hash of the Public Key in *mtk-dev.pem/mtk-dev384.pem/mtk-dev521.pem*

Use the following OpenSSL commands to calculate the SHA-256/SHA-384/SHA-512 hash of the public key derived from the private key files my256.pem/my384.pem/my521.pem
The lengths of sha256sum is 32bytes:

```
$ openssl pkey -inform PEM -in my256.pem -pubout -outform DER | sha256sum
0cd1705b518f10b74552626559d44ce3132552299dee6f81d3d541fcf544416e  -
```

The lengths of sha384sum is 48bytes:

```
$ openssl pkey -inform PEM -in my384.pem -pubout -outform DER | sha384sum
4ca886c7fb807cfbb4aa3a2ee3e9751e463b310a8a0677ecffbe2eda02346e57
24363f6ac71ed9d98edbac931f23b47a  -
```

The lengths of sha512sum is 64bytes:

```
$ openssl pkey -inform PEM -in my521.pem -pubout -outform DER | sha512sum
a20e0b5485ed8bfb2df592f75cf51a3f280be7e3ce92a537c65e7a6dcdb108dc
a97b9c1d723078b5af14e121e44ae8f8a856aa91986ea4536ac8f29a44407e05  -
```

3.4 Blow eFuse in the MT793X with the Hash

By completing the procedure in Section 3.3, the hash value of the public key needs to be written into the eFuse planned for it, which is the PROM/OTP hardware in the MT793X. Note that a bit in eFuse needs to be blown to enable the verification during boot up by BootROM.
The eFuse locations involved are listed below.
Sha256sum:

eFuse locations	Purpose
Group 1, 0xB0 ~ 0xBF	0cd1705b518f10b74552626559d44ce3 (1st half of hash, SBC_PUBK_HASH0)
Group 1, 0xC0 ~ 0xCF	132552299dee6f81d3d541fcf544416e (2nd half of hash, SBC_PUBK_HASH0)
Group 1, 0x100, bit 0	**1** (EFUSE_SBC_EN)

Sha384sum:

eFuse locations	Purpose
Group 1, 0xB0 ~ 0xBF	4ca886c7fb807cfbb4aa3a2ee3e9751e (1st half of hash, SBC_PUBK_HASH0)
Group 1, 0xC0 ~ 0xCF	463b310a8a0677ecffbe2eda02346e57 (2nd half of hash, SBC_PUBK_HASH0)
Group 1, 0xD0 ~ 0xDF	24363f6ac71ed9d98edbac931f23b47a (1st half of hash, SBC_PUBK_HASH1)
Group 1, 0x100, bit 0	**1** (EFUSE_SBC_EN)

Sha512sum:

eFuse locations	Purpose
Group 1, 0xB0 ~ 0xBF	a20e0b5485ed8bfb2df592f75cf51a3f (1st half of hash, SBC_PUBK_HASH0)
Group 1, 0xC0 ~ 0xCF	280be7e3ce92a537c65e7a6dcdb108dc (2nd half of hash, SBC_PUBK_HASH0)
Group 1, 0xD0 ~ 0xDF	a97b9c1d723078b5af14e121e44ae8f8 (1st half of hash, SBC_PUBK_HASH1)
Group 1, 0xE0 ~ 0xEF	a856aa91986ea4536ac8f29a44407e05 (2nd half of hash, SBC_PUBK_HASH1)
Group 1, 0x100, bit 0	**1** (EFUSE_SBC_EN)

For the procedure for how to blow eFuse, please see FBTool documentation.

WARNING: During an experiment/hands-on training, you should program SBC_PUBK_HASH0 and check whether the eFuse is correctly programmed and matches the generated key hash before moving on. Enabling secure boot with wrongly programmed key turns the MT793X device into a brick.

3.5 Blow eFuse with the Hash by FBTool

User can use FBTool to blow eFuse, below show example of blow sha256sum, the operating steps as:

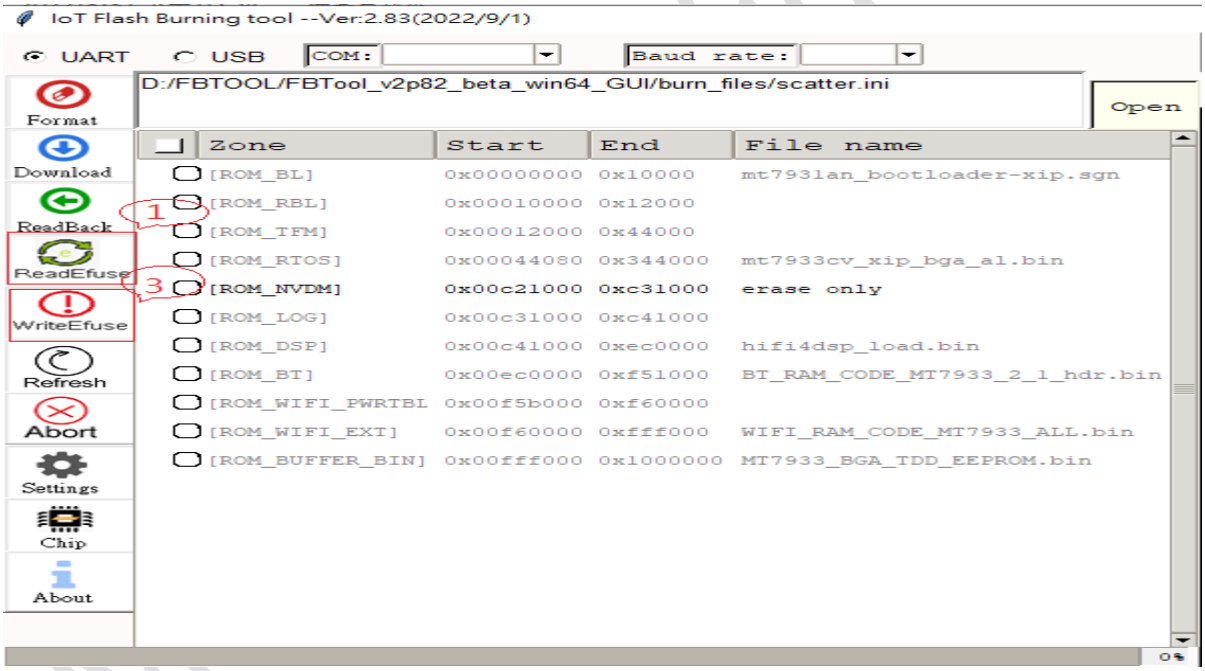


Figure 2: Blow eFuse by FBTool

Step1:

To get current efuse values of your platform by run “ReadEfuse” command, it will output customer_ef_tbl.txt.

Step2:

Edit customer_ef_tbl.txt of [SBC_PUBK_HASH0] and [SBC_EN] blocks and the content as Figure3.

Notice of the big/little endian are different as description on section 3.4. therefore, for conveniently ,SDK will output the SBC_PUBK_HASH data when build bootloader as show on Figure4.

Step3:

Blow eFuse by run “WriteEfuse” command

```
[SBC_PUBK_HASH0]
comment = security boot public key0 ,must be HEX as "0A" not "A";EDCSA_P256_SHA256:write to index11-12; EDCSA_P384_SHA384:write to index11-13; EDCSA_P521_SHA512:write to index11-14
blow_enable = y
index = 11
len = 32
data = 5b,70,d1,0c,b7,10,8f,51,65,62,52,45,e3,4c,d4,59,29,52,25,13,81,6f,ee,9d,fc,41,d5,d3,6e,41,44,f5

[SBC_PUBK_HASH1]
comment = security boot public key1 ,must be HEX as "0A" not "A"
blow_enable = n
index = 13
len = 32
data = 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00

[SBC_EN]
comment = enable secure boot.(1-enable, 0-disable)
blow_enable = y
index = 16
len = 1
data = 1
```

Figure 3: Content of enable SBC on customer_ef_tbl.txt

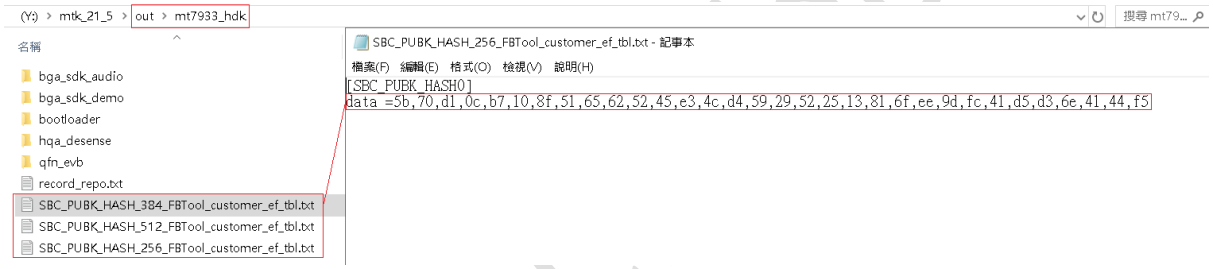


Figure 4: Output SBC_PUBK_HASH when build bootloader for FBTool

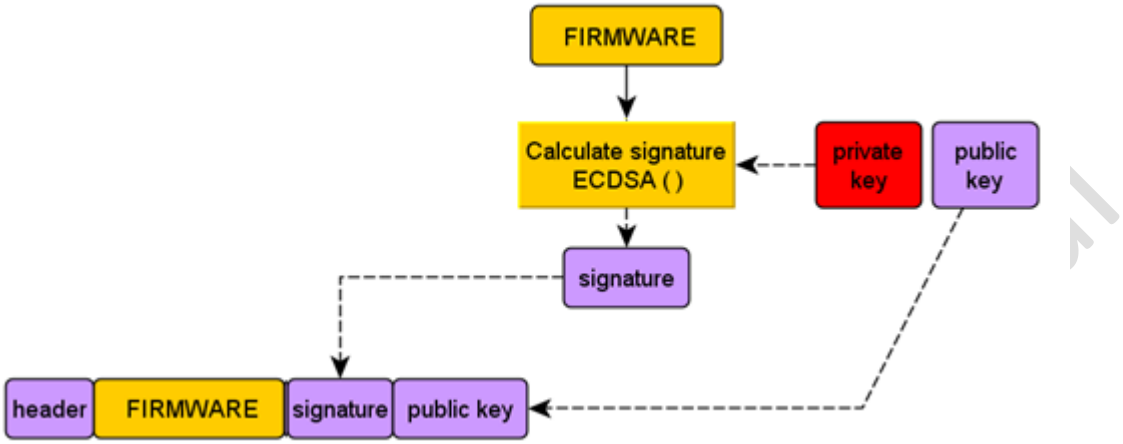
3.6 Download the Newly Built Firmware into Flash

Once the steps in Section 3.4 are fully completed, the device MT793X has secure boot enabled and the secure boot cannot be turned off. However, you can still download firmware images with FBTool and boot the system up with signed images.

For the procedure for how to download firmware images, please see FBTool documentation.

4 Developer Resources

4.1 Signature Generation Procedure



The signature generation procedure is described in Figure 4.

Figure 5. Signature Generation Procedure

The signing source code can be found in a Python-based script at *tools/mcuboot/scripts/imgtool.py* in the SDK and more of its implementation under the folder *tools/mcuboot/scripts/imgtool*.

4.2 Signature Verification Procedure

The signature verification happens at two locations in the MT793X.

The first verification happens if the eFuse bit (EFUSE_SBC_EN) is set, which indicates secure boot is enabled for BootROM. If secure boot in BootROM is enabled, BootROM verifies against BOOTLOADER.

Similarly, BOOTLOADER verifies APPLICATION firmware and possibly more firmwares. The verification happens based on the status of eFuse bit (EFUSE_SBC_EN).

The verification source code can be found in C file *project/mt7933_hdk/apps/bootloader/src/bl_image.c* and the directory *middleware/MTK/sboot*.

5 Q&A

This page is intentionally left blank as no question has been added yet.

6 References

1. SHA-256 – Description in fips180-2 on NIST website.
2. ECC P-256 - Description in P-256 on NIST website.
3. imgtool.py – imgtool is a utility from MCUBOOT. It can be found at https://mcu-tools.github.io/mcuboot/.

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.