TROPIC01

ODN_TR01_app_007

Firmware update Application Note

Version: 1.2

Abstract

This application note describes the firmware architecture of TROPIC01 and its secure update process. It serves as a guide for performing firmware updates through a Host MCU. It provides a high-level overview of security mechanisms of firmware update. It also outlines the procedure for enabling custom vendor firmware on TROPIC01.

December 5, 2025

tropicsquare

Contents

Glossary

- **MCU** – Micro-Controller Unit
- **FW** – Firmware
- **ECC** – Elliptic-curve Cryptography
- **SPECT** – Secure Processor of Elliptic Curves for TROPIC01 (used interchanbeably with '**ECC engine**' in this document)
- **ROM** – Read-only Memory
- **SDK** – Software Development Kit
- **I-Memory** – Irreversible Memory
- **R-Memory** – Reversible Memory
- **OTP** – One Time Programmable memory
- **CPU** – Central Processing Unit - a RISC-V CPU in case of TROPIC01
- **API** – Application Programming Interface

1 Introduction

The TROPIC01 runs a mutable (updatable) firmware (FW). The FW can be updated via a Host MCU. The firmware update and the FW boot are both secured against running unauthenticated FW. The FW may be signed either by Tropic Square, or by a customer (see Sec. 4).
Newer FW versions may introduce new functionality, or provide functional or security bug-fixes. It is therefore important that the system integrator implements TROPIC01 FW update in devices operating in the field.

1.1 Document structure

2. Functional description:
Explains how the TROPIC01 FW operates, boots and is updated.

3. FW update process:
How to perform the FW update via Libtropic [5], the TROPIC01 SDK, with a code example.

4. Custom FW introduction process:
Explains how a customer can sign TROPIC01 FWs by himself and what are the benefits and implications.

Note

User is shielded from the implementation details given in Section 2 via the TROPIC01 SDK – Libtropic. You may refer directly to Section 3 to see how to use Libtropic to update the TROPIC01 FWs.

2 Functional description

This section explains how the TROPIC01 secure upload (update) and boot processes function.

TROPIC01 contains the following FW execution engines:

- RISCV CPU
- ECC engine (SPECT)

Note

'ECC engine' and 'SPECT' are used interchangeably.

There are multiple kinds of FW running in TROPIC01:

- **Immutable FW (bootloader)** - located in ROM, runs on RISCV CPU from ROM after power-up, updates or boots the mutable FWs
- **RISCV Mutable FW (CPU FW)** - updatable, located in R-memory, runs on RISCV CPU from RAM, processes L2/L3 communication
- **ECC engine mutable FW (ECC engine FW)** - updatable, located in R-memory, runs on ECC engine from RAM, helps the CPU FW with processing ECC commands (***ECC_Key_****, ***ECDSA/EDDSA_Sign***).

After power-up, the TROPIC01 is transiently in Start-up mode (bootloader is validating and booting mutable FWs). On successful boot, TROPIC01 transfers to Idle mode. To perform a FW update, TROPIC01 must stay in Start-up mode. To achieve staying in Start-up Mode, the Host MCU issues ***Startup_Req*** with **MAINTENANCE_REBOOT** (see [1]).
The bootloader is hard-wired in TROPIC01 ROM since manufacturing and contains a *bootloader public key*. A *bootloader private key* is in possession of Tropic Square. The bootloader and the *bootloader public key* together form an immutable Root-of-Trust of the boot chain.
TROPIC01's I-memory contains a *vendor public key* which is signed by the *bootloader private key*. The bootloader on each power-up first verifies the signature of the *vendor public key*.
TROPIC01 FWs (the CPU and ECC engine FWs) are authenticated and checked for integrity via digital signatures. The FWs are signed by the *vendor private key*. An entity who posseses the *vendor private key* can release FW packages that will boot on TROPIC01 with the coresponding *vendor public key*.

The said entity is usually Tropic Square. A customer can order a special TROPIC01 batch that contains a *vendor public key* whose *vendor private key* is customer's secret. See Section 4 for more details.

Note

Which *vendor public key* is in a given TROPIC01 chip is visible from its part number (see [3]). Customer can verify the part number via the on-chip X.509 certificate (see [4]).

The following diagram shows TROPIC01 boot chain and location of its components:

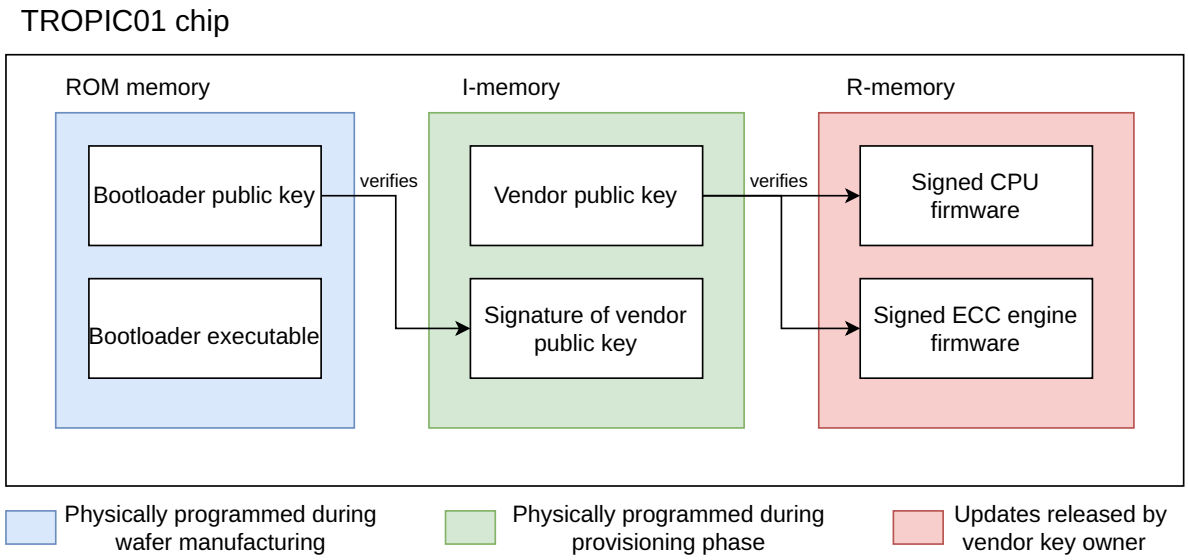

Figure 1: Storage of keys and firmware in TROPIC01.

Authenticity and integrity of TROPIC01 FWs is checked (by digital signature verification):

- during FW update (upload) - see Section 2.1
- during every boot - see Section 2.2

The CPU FW and the ECC engine FWs are updated separately, but their update process is identical. For each of the FWs, there are 2 banks in R-memory, where the FWs can be stored. The bootloader chooses automatically which bank is used for boot and which for update (details follow in the next Sections).
See Figure 2 for the data path of the FW during update and boot:

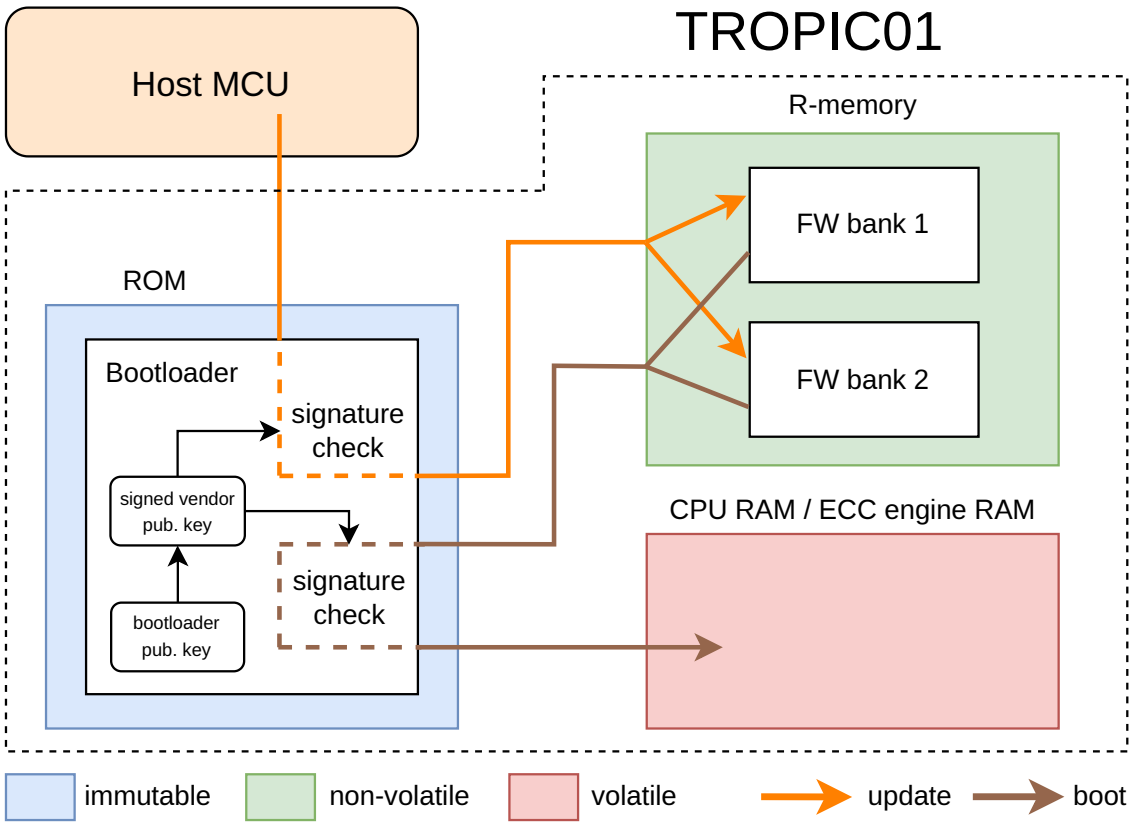

Figure 2: Data path of TROPIC01 CPU and ECC engine (SPECT) FWs.

2.1 Authenticated FW upload

There are 2 banks in R-memory for each FW type (CPU and ECC engine FW). The bank that is booted is called the **protected** bank. The other bank is called **unprotected** and is the target of the update process.
The banks are defined on each chip restart. The following table shows how the bootloader selects the banks:

Bank 1 state	Bank 2 state	protected bank	unprotected bank
valid FW	*invalid*	bank 1	bank 2
invalid	*valid* FW	bank 2	bank 1
valid newer FW	*valid* older FW	bank 1	bank 2
valid older FW	*valid* newer FW	bank 2	bank 1
same *valid* FW	same *valid* FW	bank 1	bank 2
invalid	*invalid*	bank 1	bank 2

Note

In the last case, TROPIC01 stays in Start-up mode (does not boot any FW since there is no valid FW).

Note

For an exact definition of a *valid* and *invalid* bank state, see Section 2.2

The FW upload mechanism uses a signature check early in the process. This ensures that only authenticated and integrity-verified FW is ever written to R-memory of TROPIC01. To achieve these security guarantees, the FW binary is split into chunks that create a signed hash chain.
The first chunk (FW chunk 0) contains a FW type (CPU/ECC engine), a version of the FW and a hash of the next chunk. This data is digitally signed by the *vendor private key* and the signature is also included in the initial chunk. All other chunks contain a part of the FW binary with a hash of the following chunk.

See Figure 3 for the signed chunked hash chain containing the FW update:

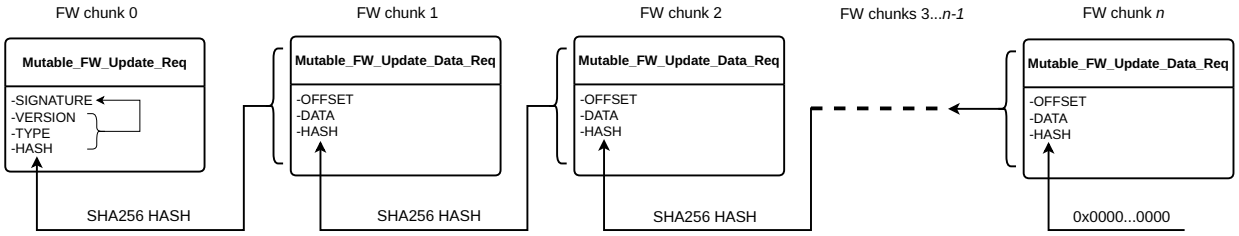

Figure 3: FW signed chain of chunks for update.

The Host MCU sends the chunks one-by-one to TROPIC01. For chunk 0, the bootloader verifies the signature using the *vendor public key*.
If the FW version of the incoming update is lower than the FW version in the **protected** bank, the bootloader rejects the update. This prevents FW downgrade. The bootloader then trusts and saves the hash of the next block.
For all following chunks after chunk 0, the saved hash is verified against the hash of the receiving chunk. On success, the FW chunk is written to R-memory. See Figure 4 for the bootloader logic during FW upload process:

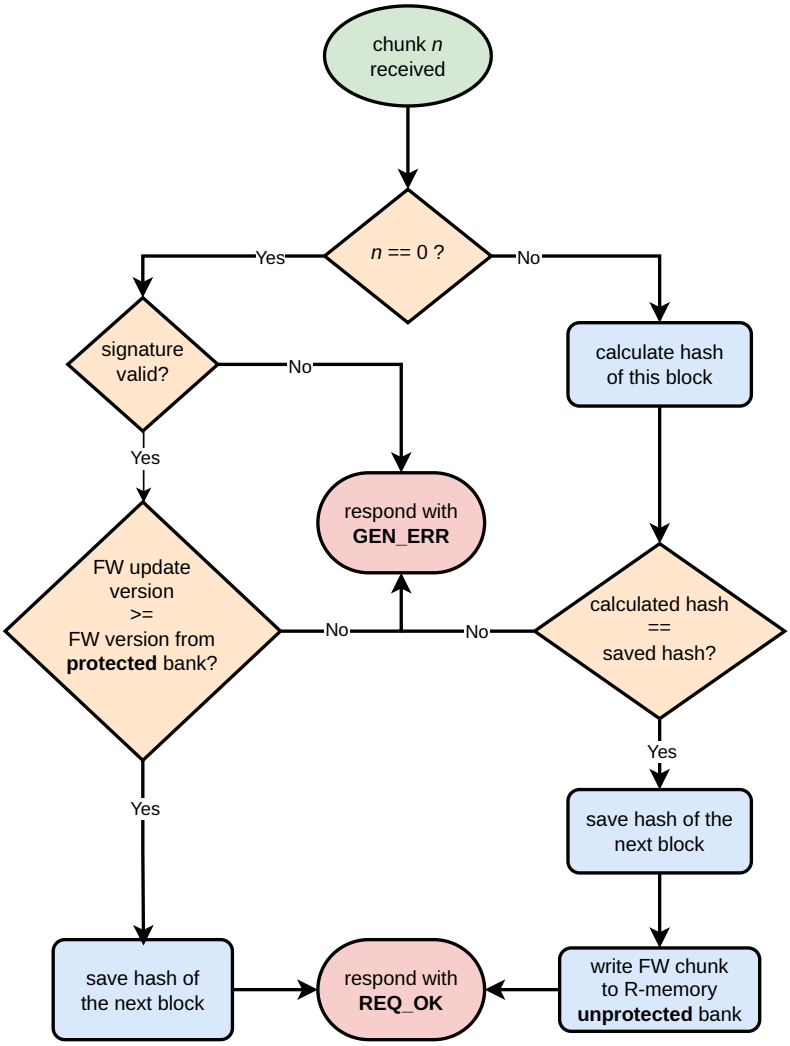

Figure 4: Bootloader FW upload process flow chart.

Note

The TROPIC01 FWs are not encrypted. This is a feature. Anyone can verify that the FW binary signed by Tropic Square that is used for the update matches the source code (see the source at [7, 8]).
The same can be done for any entity that has its *vendor public key* inside TROPIC01 if the source code is provided. Binary reverse-engineering can be done even without the source code.

2.2 FW boot authentication

After power-up, TROPIC01 is transiently in Start-up mode. In this mode, the bootloader verifies the authenticity and integrity of both CPU and ECC engine FWs from the **protected** banks and boots them. On success, TROPIC01 transitions to Idle mode. On error, TROPIC01 stays in Start-up mode.
The FW banks have the following structure:

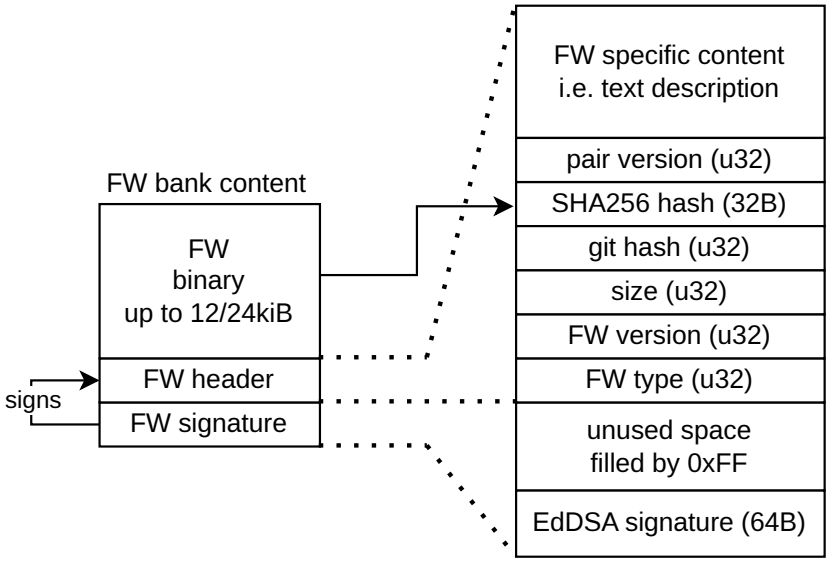

Figure 5: Structure of the FW bank in R-memory.

The versions of the CPU and ECC engine FWs must match. This is enforced by a *pair version* field in the FW header. Each CPU FW contains a version of the ECC engine FW that it requires in the FW header. Bootloader during each boot checks the *pair version*. On mismatch of the *pair version*, TROPIC01 stays in Start-up mode.

! Warning !

It is the responsibility of the Host MCU to always update matching CPU and ECC engine (SPECT) FWs. See the compatibility table for matching versions in [5] GitHub.

During Start-up mode, bootloader verifies the TROPIC01 Mutable FWs in both banks in the following way:

1. Load the FW header and signature from the FW bank
2. Check the FW header for valid information (*type*, *version* and *size* fields make sense)
3. Check (verify) the EdDSA signature of the FW header
4. Calculate the SHA256 hash of the FW binary from the FW bank and compare to the *hash* field of the FW header

If any of the checks fail, the bootloader considers that FW bank *invalid*. Otherwise it is considered *valid*. The bootloader then chooses the **protected** and **unprotected** banks (see table in Section 2.1). If both FW banks are *invalid*, TROPIC01 stays in Start-up mode.

Bootloader then proceeds to boot the FW from the **protected** bank:

1. Copy FW binary from the FW bank to RAM
2. Calculate the SHA256 hash of the FW binary from RAM and compare to the *hash* field of the FW header
3. When verifying CPU FW:
   - save the *pair version* field.

   When verifying ECC engine FW:
   - check the saved *pair version* matches the *FW version* field.

At the end of this process, TROPIC01 transitions to Idle mode and is running the mutable FWs.
More details about the FW update and boot process mechanism and API can be found in [1, 2, 5].

3 FW update process

This section describes how to update TROPIC01 FWs using the TROPIC01 SDK – Libtropic [5].
TROPIC01 FW update files are bundled together with Libtropic. We recommend the following:

1. In the main README of the Libtropic repository [5], see the compatiblity table for a list of compatible versions of Libtropic, CPU and ECC engine FW. TROPIC01 will stay in Start-up mode unless compatible versions of both FWs are uploaded. We recommend to always get the latest available version of Libtropic.
2. See the *Get Started* → *TROPIC01 Firmware* section in the Libtropic documentation [5], which provides information on how to work with the TROPIC01 FW in Libtropic.

Note

If some other entity is in charge of releasing FW updates to given TROPIC01 chips (the chips contain other than Tropic Square *vendor public key*, see Section 4), the FW update files (and compatibility table) have to be obtained from the said entity.

A step-by-step high-level process for updating TROPIC01 FWs is the following:

1. Select suitable and compatible versions of CPU and ECC engine FWs.
2. Perform maintenance restart of TROPIC01by sending ***Startup_Req*** with **MAINTENANCE_REBOOT**.
3. Check the chip stayed in Start-up mode.
4. Update the CPU FW.
5. Update the ECC engine FW.
6. Perform maintenance restart of TROPIC01by sending ***Startup_Req*** with **MAINTENANCE_REBOOT**.
7. Check the chip stayed in Start-up mode.
8. Update the CPU FW – second FW bank will be updated.
9. Update the ECC engine FW – second FW bank will be updated.
10. Check all FW banks contain the new FW versions.

11. Perform a regular restart of TROPIC01 by sending ***Startup_Req*** with **REBOOT** and check the desired FW version has been booted.

Note

We strongly recommend to always update the latest TROPIC01 FWs to both FW banks to reduce the possibility of a downgrade attack.

As explained in the *Get Started* → *TROPIC01 Firmware* section in the Libtropic documentation [5], Libtropic provides the TROPIC01 FW update files in two formats:

1. C header files (`*.h`). These are designed to be included and compiled directly into the Host MCU's firmware/application.
2. Binary files (`*.bin`). These can be stored in the Host MCU's filesystem or external storage, loaded at runtime and used to update TROPIC01's FW.

The process listed above is implemented in Libtropic's code example `lt_ex_fw_update`. The latest version of this example is available in the Libtropic repository [6] and it is documented in the Libtropic documentation [5]. We also provide the example's code below. The example can be used as a basis when integrating the TROPIC01 FW update process in your own application. However, note that the example uses C header file format for the FW update files.

This is the source code of the aforementioned Libtropic's `lt_ex_fw_update` example:

```
int lt_ex_fw_update(lt_handle_t *h)
{
    LT_LOG_INFO("====================================");
    LT_LOG_INFO("==== TROPIC01 FW update Example ====");
    LT_LOG_INFO("====================================");

    lt_ret_t ret;
    LT_LOG_INFO("Initializing handle");
    // Note: It is assumed that the ‘h.l2.device‘ and ‘h.l3.crypto_ctx‘ members were already
    // initialized. Because these members are pointers, the assigned structures must exist
    // throughout the whole life-cycle of the handle.
    // Refer to the 'Get Started'->'Integrating Libtropic'->'How to Use' Section in the
    // Libtropic documentation for more information.
    ret = lt_init(h);
    if (LT_OK != ret) {
        LT_LOG_ERROR("Failed to initialize handle, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }

    // The chip must be in Start-up Mode to be able to perform a firmware update.
    LT_LOG_LINE();
    LT_LOG_INFO("1. Sending maintenance reboot request");
    ret = lt_reboot(h, TR01_MAINTENANCE_REBOOT);
    if (ret != LT_OK) {
        LT_LOG_ERROR("lt_reboot() failed, ret=%s", lt_ret_verbose(ret));
```

```
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_LINE();
    LT_LOG_INFO("2. Updating TR01_FW_BANK_FW1 and TR01_FW_BANK_SPECT1");
    LT_LOG_INFO("2.1. Updating RISC-V FW");
    ret = lt_do_mutable_fw_update(h, fw_CPU, sizeof(fw_CPU), TR01_FW_BANK_FW1);
    if (ret != LT_OK) {
        LT_LOG_ERROR("RISC-V FW update failed, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_INFO();
    LT_LOG_INFO("2.2. Updating SPECT FW");
    ret = lt_do_mutable_fw_update(h, fw_SPECT, sizeof(fw_SPECT), TR01_FW_BANK_SPECT1);
    if (ret != LT_OK) {
        LT_LOG_ERROR("SPECT FW update failed, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_LINE();
    LT_LOG_INFO("3. Updating TR01_FW_BANK_FW2 and TR01_FW_BANK_SPECT2");
    LT_LOG_INFO("3.1. Updating RISC-V FW");
    ret = lt_do_mutable_fw_update(h, fw_CPU, sizeof(fw_CPU), TR01_FW_BANK_FW2);
    if (ret != LT_OK) {
        LT_LOG_ERROR("RISC-V FW update failed, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_INFO();
    LT_LOG_INFO("3.2. Updating SPECT FW");
    ret = lt_do_mutable_fw_update(h, fw_SPECT, sizeof(fw_SPECT), TR01_FW_BANK_SPECT2);
    if (ret != LT_OK) {
        LT_LOG_ERROR("SPECT FW update failed, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_LINE();
    LT_LOG("Successfully updated all 4 FW banks:");

    ret = lt_print_fw_header(h, TR01_FW_BANK_FW1, printf);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to print TR01_FW_BANK_FW1 header, ret=%s", lt_ret_verbose(ret))
    ;
        lt_deinit(h);
        return -1;
    }
    ret = lt_print_fw_header(h, TR01_FW_BANK_FW2, printf);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to print TR01_FW_BANK_FW2 header, ret=%s", lt_ret_verbose(ret))
    ;
        lt_deinit(h);
        return -1;
```

```
    }
    ret = lt_print_fw_header(h, TR01_FW_BANK_SPECT1, printf);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to print TR01_FW_BANK_SPECT1 header, ret=%s", lt_ret_verbose(
    ret));
        lt_deinit(h);
        return -1;
    }
    ret = lt_print_fw_header(h, TR01_FW_BANK_SPECT2, printf);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to print TR01_FW_BANK_SPECT2 header, ret=%s", lt_ret_verbose(
    ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_LINE();

    LT_LOG_INFO("Sending reboot request");
    ret = lt_reboot(h, TR01_REBOOT);
    if (ret != LT_OK) {
        LT_LOG_ERROR("lt_reboot() failed, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK, TROPIC01 is executing Application FW now");

    LT_LOG_LINE();
    LT_LOG_INFO("Reading RISC-V FW version");
    // This variable is reused on more places in this block to store different FW versions
    uint8_t fw_ver[TR01_L2_GET_INFO_RISCV_FW_SIZE] = {0};
    ret = lt_get_info_riscv_fw_ver(h, fw_ver);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to get RISC-V FW version, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_INFO("RISC-V FW version: %02" PRIX8 ".%02" PRIX8 ".%02" PRIX8 " (+ .%02" PRIX8 ")
    ", fw_ver[3], fw_ver[2], fw_ver[1], fw_ver[0]);

    LT_LOG_INFO();
    LT_LOG_INFO("Reading SPECT FW version");
    ret = lt_get_info_spect_fw_ver(h, fw_ver);
    if (ret != LT_OK) {
        LT_LOG_ERROR("Failed to get SPECT FW version, ret=%s", lt_ret_verbose(ret));
        lt_deinit(h);
        return -1;
    }
    LT_LOG_INFO("OK");

    LT_LOG_INFO("SPECT FW version: %02" PRIX8 ".%02" PRIX8 ".%02" PRIX8 " (+ .%02" PRIX8 ")"
    , fw_ver[3], fw_ver[2], fw_ver[1], fw_ver[0]);

    LT_LOG_LINE();
    LT_LOG_INFO("Deinitializing handle");
    ret = lt_deinit(h);
    if (LT_OK != ret) {
        LT_LOG_ERROR("Failed to deinitialize handle, ret=%s", lt_ret_verbose(ret));
        return -1;
    }
    return 0;
}
```

4 Custom FW introduction process

TROPIC01 supports transfering ownership of FW update releases to a different entity. TROPIC01 then accepts FW updates coming only from that entity.
Tropic Square allows customers to sign the mutable FWs by themselves. Such FWs will run only on specific TROPIC01 devices that contain customers *public vendor key*.
There are two possible scenarios:

- **Standard Scenario:** Tropic Square owns the *vendor private key* corresponding to the *vendor public key* in TROPIC01. This ensures that only Tropic Square can release FW updates for such TROPIC01 chips.
- **Alternative Scenario:** Customer owns the *vendor private key* and delivers the corresponding *vendor public key* to Tropic Square. Tropic Square writes the *vendor public key* to TROPIC01 during manufacturing. The manufactured chips can boot only the mutable FWs signed by the customer.

The alternative scenario is useful for:

- releasing of custom FW, which might contain new functionality or remove unused functionality.
- re-releasing of the official Tropic Square FW. The releasing entity is in charge of building the FWs from source and therefore does not have to trust the provided FW binaries. Furthermore, this enables 'accepting' the official Tropic Square FW update by signing it. Unless signed by *vendor private key*, the FW would not be accepted by TROPIC01.

Note

TROPIC01 chips with different programmed *vendor public key* will have a different part number assigned. If you would like to order TROPIC01 with your own *vendor public key*, contact Tropic Square for more details.

The process of introducing a new *vendor public key* into TROPIC01 is depicted in Fig. 6. The generated *vendor private key* has to be kept secured. Anyone with *vendor private key* is able to upload a potentially malicious FW to TROPIC01 with the given *vendor public key*. A leak of *vendor private key* would undermine the security guarantees of the product. Tropic Square is not responsible for the FW running on TROPIC01 with different than the Tropic Square *vendor public key*.

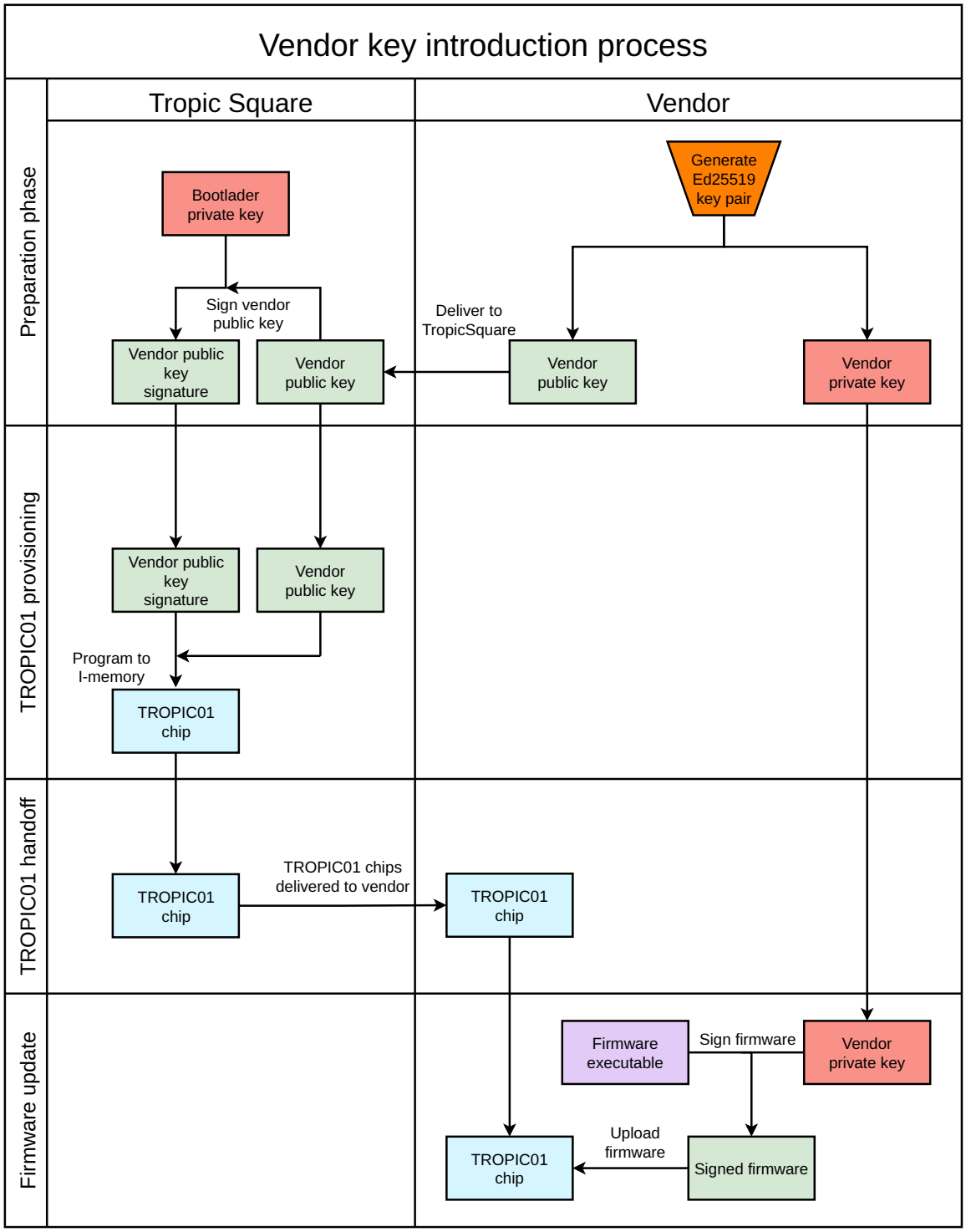

Figure 6: Vendor key introduction process flow.

4.1 Creating FW update files

Customer with a TROPIC01 with non-Tropic Square *vendor public key* has to create the FW update files himself.
Creating the FW update files from the compiled binary requires:

1. attaching a header with a signature to the FW binary (as described in Sec 2.2)
2. creating the signed hash chain (as described in Sec 2.1).

For this purpose, Tropic Square provides a 'fw_packager' tool [9].
The tool supports 3 operations: 'pack', 'sign' and 'chain'. You can see help for each of the operations:

```
python3 fw_packager.py {pack,sign,chain} -h
```

Pack

Packages the compiled FW binary - adds a header and a signature.

```
python3 fw_packager.py pack
    --inp FW_BINARY  # .hex32 binary
    --ver VERSION    # for example 'v1.2.3'
    --type TYPE      # 'APP'/'SPECT'
    --git-hash HASH      # 32bit FW revision, for example a part of git commit hash ('2
    bca14bf')
    --key PRIV_KEY   # Ed25519 private key in PEM format
    --out OUT_FILE
```

Note

'APP' type corresponds to the RISCV CPU FW, 'SPECT' type corresponds to the ECC engine FW.

Sign

Signs the already packed binary (from the 'pack' operation). Replaces the signature with a new one from the provided private key.

```
python3 fw_packager.py sign
    --inp PACKED_FW_BINARY
    --key PRIV_KEY   # Ed25519 private key in PEM format
    --out OUT_FILE
```

Note

The 'sign' operation is useful for splitting the pack operation from the signing. In this case, perform the 'pack' operation with a dummy key and use the real key for the 'sign' and 'chain' operations.

Chain

Creates a signed hash chain from the packed binary. The resulting file is the FW update file that can be used to update TROPIC01 FW.

```
python3 fw_packager.py chain
    --inp PACKED_FW_BINARY
    --key PRIV_KEY   # Ed25519 private key in PEM format
    --out OUT_FILE
```

The 'fw_packager' tool is available on Tropic Square GitHub, see [9].

4.1.1 Generate vendor key

You can generate the *vendor private key* and *vendor public key* in the PEM format for example via OpenSSL:

```
openssl genpkey -algorithm ed25519 -out private.pem
openssl pkey -in private.pem -pubout > public.pem
```

The 'private.pem' and 'public.pem' files contain the *vendor private key* and the *vendor public key* respectively.
The customer shall generate the vendor keypair only once and he shall keep the *vendor private key* secret. After delivering the generated *vendor public key* in the PEM format to Tropic Square, the customer can order TROPIC01 with his *vendor public key* (see Figure 6).

4.1.2 Sign FW binary

To sign the compiled FW binary, you need:

1. *vendor private key* – to generate it, see 4.1.1
2. compiled FW binaries – you can modify and compile the official Tropic Square CPU FW [7] and ECC engine FW [8]
3. 'fw_packager' tool [9]

To sign the FW, execute:

```
python3 fw_packager.py pack
    --inp      spect_app-v1.0.0.hex32         # example compiled binary FW in hex format
    --type     SPECT                          # use 'APP' for CPU FW
    --key      private.pem                    # vendor private key file
    --git-hash 4507dd4                        # example version hash
    --ver      v1.0.0                         # example version
    --out spect_app-v1.0.0_signed.bin

python3 fw_packager.py chain
    --inp spect_app-v1.0.0_signed.bin
    --key private.pem                   # vendor private key file
    --out spect_app-v1.0.0_signed_chunks.bin
```

The 'spect_app-v1.0.0_signed_chunks.bin' is the resulting FW update file. With this file, you can perform a FW update directly using Libtropic if your Host Platform supports a filesystem. Just read the binaries and save the read data into C arrays and use them similarly as the arrays `fw_CPU` and `fw_SPECT` are used in aforementioned `lt_ex_fw_update` example.
If you want to use the C header format for the FW update files, see Section 4.1.3.

4.1.3 Embed the compiled FW into Libtropic

You can convert the TROPIC01 FW binaries into a C header format, that can be compiled directly into Libtropic.
This is the procedure:

1. Clone the Libtropic repository [5] and enter its `TROPIC01_fw_update_files/` directory.
2. Copy the CPU FW and ECC engine FW binaries into the `boot_v_<X_Y_Z>/fw_v_<A_B_C>/` directory (fill in the correct Bootloader and CPU FW version).
3. Run the converter script to generate the C header files from the FW binaries:

```
python3 convert.py boot_v_<X_Y_Z>/fw_v_<A_B_C>/
```

4. in Libtropic's CMakeLists.txt (located in its root directory), add your new version here:

```
set_property(CACHE LT_CPU_FW_UPDATE_DATA_VER PROPERTY STRINGS "<A_B_C>" "other
versions")
```

5. And finally, compile Libtropic. To select which CPU FW version will be compiled into Libtropic, set the Libtropic's CMake option `LT_CPU_FW_UPDATE_DATA_VER` to `"<A_B_C>"` (fill in the correct CPU FW version) during the build phase. For more information about this, refer to the *Get Started* → *Integrating Libtropic* section in the Libtropic documentation [5].

Note

The `converter.py` script expects to find a FW type (CPU/SPECT) and the FW version in the FW update file filename. For example: `spect_app-v1.0.0_signed_chunks.bin` and `cpu_app-v1.0.0_signed_chunks.bin`.

Now, your FW update files are compiled directly into Libtropic and you can use the Libtropic's aforementioned `lt_ex_fw_update` example to perform the FW update of TROPIC01.

Version history

Version	Date	Description
1.0	30.09.2025	Initial public release
1.1	06.11.2025	Change SPECT&FW GitHub link, update fw_packager example usage
1.2	27.11.2025	Sync all Libtropic parts with release v3.0.0

References

[1] ODD_TR01_datasheet, TROPIC01 Datasheet, Tropic Square

[2] ODU_TR01_user_api, TROPIC01 User API, Tropic Square

[3] OD_TR01_catalog_list, TROPIC01 Catalog list, Tropic Square

[4] ODN_TR01_app_003, TROPIC01 Device Identity and PKI, Application Note, Tropic Square

[5] TROPIC01 SDK – Libtropic,
GitHub: `https://github.com/tropicsquare/libtropic`
Documentation: `https://tropicsquare.github.io/libtropic/`

[6] libtropic FW update example,
Link: `https://github.com/tropicsquare/libtropic/blob/master/examples/lt_ex_fw_update.c`

[7] TROPIC01 RISC-V CPU firmware source code,
GitHub: `https://github.com/tropicsquare/ts-tr01-app`

[8] TROPIC01 ECC engine (SPECT) firmware source code,
GitHub: `https://github.com/tropicsquare/ts-spect-fw`

[9] fw_packager – Tool that creates a FW update file package from a FW binary.
GitHub: `https://github.com/tropicsquare/fw-packager`

Legal Notice

Our mission is to provide you with high quality, safe, and transparent products, but to be able to do so, we also have to make the following disclaimers.

To verify the characteristics of our products, consult the repositories we make available on our GitHub. While we do our best to keep the content of these repositories updated, we cannot guarantee that the content of these repositories will always identically correspond to our products. For example, there may be delays in publication or differences in the nature of the software solutions as published and as included in the hardware products. Some parts of our products cannot be published due to third party rights.

We take pride in publishing under open-source license terms, but do not grant licenses to any of our patents. Please consult the license agreement in the repository. We reserve the right to make changes, corrections, enhancements, modifications, and improvements to our products, published solutions and terms at any time without notice.

Since we cannot predict what purposes you may use our products for, we make no warranty, representation, or guarantee, whether implied or explicit, regarding the suitability of our products for any particular purpose.

To the maximum extent permitted by applicable law, we disclaim any liability for any direct, indirect, special, incidental, consequential, punitive, or any other damages and costs including but not limited to loss of profit, revenue, savings, anticipated savings, business opportunity, data, or goodwill regardless of whether such losses are foreseeable or not, incurred by you when using our products. Further, we disclaim any liability arising out of use of our products contrary to their user manual or our terms, their use/implementation in unsuitable environments or ways, or for such use which may infringe third party rights. Notwithstanding the above, the maximum liability from the use of our products shall be limited to the amount paid by you as their purchase price.