

Flyruit M10R5: restoring ADB access

Technical work report · October 16–17, 2026

Result

We established authenticated USB ADB access to a Flyruit M10R5 digital photo frame by adding the Linux computer's public ADB key to the device's boot ramdisk. The frame booted normally afterward and accepted the matching private key. A full internal-storage backup was made and verified before the boot partition was modified.

The resulting connection provides Android's standard shell account, UID 2000. Root access was not established. Frameo remains installed; changing the launcher or installing a replacement application was not part of the completed work.

Device identification

Item	Confirmed value
Retail model	Flyruit M10R5
Android product brand / manufacturer	SOMY / Allwinner
Processor	Allwinner A33, FEL SoC identifier 00001667
Actual Android version	6.0.1, SDK 23
Android build	M10R5_20260617
Android platform	astar_xr819 / astar-xr819
Security patch reported by firmware	2018-08-05

Item	Confirmed value
Frameo version as transcribed during troubleshooting	v.132.16 (80320)
Internal eMMC capacity	31,272,730,624 bytes; 61,079,552 sectors
Normal USB identifier observed	1f3a:1009
FEL USB identifier	1f3a:efe8
Host	Debian Linux, hostname elara; commands ultimately run as root

The device was initially described as Android 7. Reading its own `build.prop` from the verified backup established that it runs Android 6.0.1. The Frameo version above is retained exactly as supplied and was not independently read from its APK.

Initial problem and unsuccessful checks

Frameo's interface restricted access to Android settings and other applications. Its Beta Program and ADB Access switches were enabled, but `adb` devices consistently reported serial `9e70acd7b6a700000000` as unauthorized. No USB debugging approval dialog appeared.

We tried toggling Frameo's ADB and beta switches, reconnecting USB, restarting ADB, power-cycling the frame, and starting ADB with a fresh host key in a temporary home directory. These attempts did not produce an authorization prompt. Network ADB at `192.168.1.107:5555` returned "Connection refused."

A report about another Allwinner Frameo device suggested that missing SystemUI could prevent the approval dialog. Inspection of this M10R5 found `SystemUI.apk` and its compiled code under `/system/priv-app/SystemUI`. Therefore, absence of SystemUI was not established here. The precise reason the dialog failed remains unresolved.

Entering FEL and reading internal storage

The normal USB identifier established the Allwinner vendor family. A spare 7.44 GiB microSD card was positively identified as `/dev/sdg` on the host. Its MBR partition began at sector 8192, leaving space before the filesystem for a boot stub.

We built `sunxi-fel` from linux-sunxi's `sunxi-tools`, pinned to commit:

```
text d7bbd172a5da601a08f94479de308c6fb714a19a
```

The 8,192-byte `fel-sdboot.sunxi` stub had the expected SHA-256:

```
text
2ccec08a026840571f3804671ba72afae840db27112a43f2b1f122075bc808b
c
```

It was written to the spare card at an 8 KiB offset. Booting the frame with that card inserted produced USB identifier `1f3a:efe8`. The processor query returned:

```
text AWUSBFEX soc=00001667(A33) 00000001 ver=0001 44 08
scratchpad=00007e00 00000000 00000000
```

Next, we built the A33 backup U-Boot configuration from the `wesbos/photo-frame-dashboard` project using Docker. The build used U-Boot 2025.10, DRAM clock 456 MHz, ZQ value 15291, and read-only USB mass-storage support.

The first main-loader attempt timed out after executing the SPL, the component that initializes memory. Completely removing both USB and external power restored responsive FEL. Running the supplied warm-up SPL before the main loader resolved the timeout. U-Boot and its scripts were loaded into RAM over USB.

The initial read-only mass-storage session exposed the microSD as one USB disk and the eMMC as another. Linux reported the eMMC as read-only. Automatically mounted partitions were unmounted before backup work.

Backup and verification

The complete eMMC user area was copied to this path on the Debian computer:

```
text /home/misch/frame-backup/flyruit-m10r5-original.img
```

The file size was verified as exactly **31,272,730,624 bytes**. A complete byte comparison against the exposed eMMC returned exit code **0**. This established that the backup matched the accessible storage at the time of backup.

The backup was attached as a read-only loop device. Its system partition, `/dev/loop0p7`, was mounted with `ro,no-load` to prevent ext4 journal replay. The boot partition was identified from the partition table as partition 6, beginning at sector **172032**, with a size of **32768 sectors** (16 MiB).

The partition table declares a data-partition end beyond the physical device capacity. This existing layout was preserved. The backup covers the actual device capacity; no attempt was made to extend or repair the filesystem.

Boot and authorization findings

System properties included:

```
text persist.sys.usb.config=adb ro.adb.secure=1
persist.service.adb.enable=1
```

The boot ramdisk's `default.prop` included:

```
text ro.secure=1 ro.debuggable=0 persist.sys.usb.config=none
```

The USB initialization file contained working ADB service triggers for several USB modes. In particular, `ptp`, `adb` corresponded to the originally observed USB product ID 1009.

There was no existing `adb_keys` file in the ramdisk. The device's own `sbin/adbd` binary contained the `/adb_keys` path. Together with the existing authorization requirement, these findings supported adding a trusted public key to the boot ramdisk.

Modification prepared and applied

We used the computer's normal ADB key pair under `~/.android/`, in the root account used for the final commands. Only the public key was read into the patch. The private key was never supplied to the frame or included in the report.

A local adaptation of the repository's boot patcher was saved on the Debian host as:

```
text /root/photo-frame-dashboard/scripts/a33-uboot/m10r5-key-patch.py
```

The adapted patcher added a root-owned, mode `0644` file named `adb_keys` to the boot ramdisk. It retained the existing ramdisk entries and kernel, recompressed the archive within its original allocated slot, and updated the boot header's ramdisk size and image ID.

The patcher re-parsed and checked the result. It verified that the public key was present, existing archive entries remained unchanged, the kernel was unchanged, header differences were limited to the expected fields, and bytes beyond the ramdisk slot were unchanged.

Patch payloads, a write script, a restore script, and a plan were generated under:

```
text /root/photo-frame-dashboard/diagnostics/m10r5-key-patch/
```

The generated write script checked payload CRC-32 values and compared the original boot bytes on eMMC against the verified backup. It wrote the ramdisk first, read it back and compared it, then wrote the header. The

frame was returned to FEL, the warm-up SPL was run, and U-Boot loaded the payloads and write script into RAM before applying them.

Modified region	Start sector	Sector count	Byte offset	Byte count
Boot header	172032	2	88,080,384	1,024
Boot ramdisk slot	191892	9264	98,248,704	4,743,168

These changes added a trusted host key. We did not disable ADB authentication or change `ro.secure` or `ro.debuggable`.

Verification after writing

The write script re-exposed the eMMC as a read-only USB disk. At that point Linux assigned it `/dev/sdh`; earlier it had been `/dev/sdj`. Device letters were checked again rather than treated as permanent identifiers.

Both written ranges were independently compared against their generated payloads:

```
text Header verification: 0 Ramdisk verification: 0
```

These checks verified the modified ranges. A second full-device comparison after patching was not performed.

Power and USB were disconnected, the microSD card was removed, and the frame was allowed to boot its internal Android installation. The ADB connection then accepted the computer's key. Running `adb shell id` returned:

```
text uid=2000(shell) gid=2000(shell)
groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_
rw), 1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),
3006(net_bw_stats) context=u:r:shell:s0
```

This is direct evidence of a working, authenticated Android shell.

Future access and recovery

The frame now trusts the public key embedded in the modified boot image. Future connections using that trust entry require the corresponding private key. Preserve both `adbkey` and `adbkey.pub` from the Linux account used during this work. The private key must remain confidential. Another computer can use the same pair, although separate authorized keys would be preferable if normal authorization can be restored.

The original full backup and generated original header and ramdisk payloads should be retained with the patch and restore scripts. The restore script was generated but has not been executed or tested. Recovery would require returning to FEL, reinitializing RAM, loading the appropriate payloads and guarded restore script, and verifying the restored bytes. Do not write a whole backup to a guessed Linux disk name.

A firmware update that replaces the boot image may remove the embedded public key. The successful patch does not establish that future firmware versions will accept the same procedure.

The FEL microSD remains a boot-access tool. Leave it out for normal Android operation.

Build records and references

Generated loader SHA-256 values recorded during this session:

```
```text u-boot-sunxi-with-spl.bin
2bee186f750601762937a3926888ac2a68868c089b55360e596e96e2a6627a
2f
```

```
sunxi-spl-warmup.bin
d1cbcc667ea358c9d3ee23e162e67f71e434b628a60375c78338070530699c
68
```

ums-readonly.scr

437a44f39fdcdc0a77c602ee8a285856fd2cc311802d0664f660fed9aaf15539

```

The photo-frame-dashboard repository revision was not recorded in the conversation. Preserve the local checkout and obtain `git rev-parse HEAD` there if a complete reproduction record is needed. The backup and final patched boot image were not assigned SHA-256 hashes during the recorded procedure; verification used byte comparisons.

References consulted during the work:

- [Frameo: How to Enable ADB on Your Frame](#)
- [linux-sunxi sunxi-tools](#)
- [A33/PF1007L identification and FEL guide](#)
- [A33 mainline U-Boot guide](#)
- [Related Frameo authorization issue, different model](#)

This report records the commands, outputs and inspections from the session. The related device guides informed the approach; their hardware and firmware assumptions were checked against this frame as work progressed. The exact cause of the missing authorization dialog remains open.