

Remove SOTI MobiControl from a Sunmi V2s Plus and Install Google Play + DoorDash (Android 11)

Tested on a Menulog-branded Sunmi V2s Plus (MT6765 / Helio P35, Android 11, firmware SP3136-sunmi-20240806). Adapted from a community guide for the Just Eat variant, with the extra steps this unit needed.

Overview

Sunmi V2s Plus tablets supplied by food-delivery platforms are locked to SOTI MobiControl (an MDM): no Developer Options, no browser, no Google Play. This guide removes MobiControl from the firmware, roots the tablet with Magisk, and adds Google Play so you can install apps such as DoorDash Order Manager.

End result: Android 11 with no MDM, Developer Options and ADB working, Magisk root, and Google Play Services, Play Store and Services Framework running as privileged system apps.

Warning. You will write raw partitions. A mistake can brick the unit. Only do this on hardware you own or are allowed to modify, and check any lease or merchant terms first. Everything here is at your own risk. Keep the full backup from Part 2 on two drives: it contains IMEI and calibration data (nvram, nvdata, proinfo, persist) that cannot be recreated.

Allow 3 to 5 hours the first time. You need a Windows 11 PC, a USB cable that carries data, and about 25 GB of free disk space.

What you need

Item	Purpose
Windows 11 PC with WSL (Ubuntu)	Edit the firmware images
mtkclient	Read and write partitions over MediaTek BROM
Android SDK Platform-Tools	ADB

Item	Purpose
Magisk APK	Root
MindTheGapps 11.0.0 arm64	Google apps
The tablet	Sunmi V2s Plus, charged above 50%

Paths below use forward slashes (for example `D:/sunmi`) because PowerShell, Python and mtkclient all accept them. In PowerShell, run local programs with a leading `./` (for example `./adb`).

Part 1: Set up the PC

1.1 Install Python, Git and mtkclient

This section uses **Command Prompt** (press the Windows key, type `cmd` , press Enter) rather than PowerShell.

Python. Download the 64-bit Python 3.11 installer from <https://www.python.org/downloads/windows/>. In the installer, tick **Add python.exe to PATH**, finish the installation, then open a new Command Prompt and check:

```
python --version
```

It should print `Python 3.11.x` .

Git. Download Git for Windows from <https://git-scm.com/download/win>, keep all the default options, then check:

```
git --version
```

mtkclient. Download the tool and install its Python dependencies (Command Prompt's `cd` wants backslashes, so this block is the one place they appear):

```
cd %USERPROFILE%\Desktop
git clone https://github.com/bkerler/mtkclient.git
cd mtkclient
python -m pip install --upgrade pip
pip install -r requirements.txt
```

On Windows, mtkclient also needs the **UsbDk** driver and the **MediaTek USB (MTK) driver**. Install them as described in the Windows section of the mtkclient README, because the exact driver steps change over time. Run every `python mtk.py ...` command from the `mtkclient` folder.

Do not run any write, erase or format command yet (`w` , `wl` , `e` , `ef` and so on). Reading commands such as `printgpt` (and `r1` , which you will use for the backup in Part 2) do not change the tablet. For now you are only installing the tool and testing the connection.

1.2 Check that the tablet is detected (read-only test)

This test only reads the partition table. Nothing is written or erased, so it is safe to repeat.

1. Power the tablet off completely and unplug the USB cable.
2. In Command Prompt run:

```
cd %USERPROFILE%\Desktop\mtkclient
python mtk.py printgpt
```

mtkclient now waits for the device.

3. Hold **Volume Up** on the tablet and, while still holding it, plug the USB-C cable into the PC. Keep holding for about 10 seconds.
4. If nothing happens, unplug, power off, and repeat with **Volume Down**, then with both volume keys.

Success looks like `CPU: MT6765/MT8768t(Helio P35/G35)` , `BROM mode detected` , a series of `Device detected` lines, and finally a `GPT Table` listing every partition (`boot_a` , `super` , `vbmeta_a` , `userdata` and so on).

If you see this message:

```
Preloader - [LIB]: Status: Handshake failed, retrying...
```

it means Windows can see the MediaTek port but mtkclient has not completed the handshake. Nothing has been written. Press `Ctrl+C` , unplug the cable, make sure the tablet is fully off, and try the other key combination. Do not run `printgpt` over and over in a loop.

If it still fails, look at what Windows sees:

1. Press Win + X and open **Device Manager**. Expand **Ports (COM & LPT)**, **Universal Serial Bus devices** and **Universal Serial Bus controllers**.

2. Power the tablet off, hold **Volume Down**, and plug in the cable.
3. Watch for a device that appears for a few seconds and then disappears, with a name like `MediaTek PreLoader USB VCOM`, `MediaTek USB Port`, `MediaTek DA USB VCOM`, `MTK USB Port` or `USB Serial Device (COMxx)`. If it vanishes too quickly to read, record a short screen video and play it back.
4. If nothing ever appears, try a different USB cable (some cables only charge), a USB port directly on the PC rather than a hub, and reinstall the UsbDk and MediaTek drivers.

At this stage do not use `dumpbrom`, `crash`, `payload`, `stage` or any other command that writes or exploits the device. First identify exactly which port the tablet exposes and get a clean `printgpt`. The rest of this guide assumes `printgpt` works.

1.4 WSL and Ubuntu

Open PowerShell as Administrator and run:

```
wsl --install -d Ubuntu
```

Reboot when asked, then create a username and password in the Ubuntu window (nothing appears while you type the password; that is normal). In Ubuntu run:

```
sudo apt update
sudo apt install -y e2fsprogs python3 unzip
debugfs -V
```

`debugfs -V` should print a version number. Your Windows D: drive is `/mnt/d` inside Ubuntu.

If `wsl --install` complains about virtualization, enable VT-x / SVM in your PC's BIOS.

1.5 ADB

Download <https://dl.google.com/android/repository/platform-tools-latest-windows.zip> and extract it so you have `C:/platform-tools`. Open PowerShell there and test with `./adb version`.

1.6 Working folders

Create `D:/sunmi/work`. In Ubuntu this is `/mnt/d/sunmi/work`. When you save a script with Notepad, set **Save as type: All files**, otherwise Windows appends `.txt` to the name.

Part 2: Back up the tablet and get it to boot

Put the tablet in BROM mode each time you run an mtkclient command: power it off completely, hold **Vol Down** (on some units Vol Up, or both volume keys), then plug in the USB cable. mtkclient prints `Device detected`.

1. Back up everything except userdata (15 to 30 minutes):

```
python mtk.py r1 D:/sunmi/backup --skip userdata
```

Check that `super.bin` is about 4 GiB and that `boot_a.bin`, `vbmeta_a.bin`, `vbmeta_system_a.bin` and `vbmeta_vendor_a.bin` exist. Copy the whole folder somewhere safe.

2. Unlock the bootloader:

```
python mtk.py da seccfg unlock
```

3. Erase userdata. Once the backup of the partitions is finished, you only need to erase userdata for the tablet to boot into Android (MobiControl is still installed at this point):

```
python mtk.py e userdata
```

4. Unplug, power on, and wait a few minutes. You get the Android home screen with MobiControl still present, no browser, no Google Play, and Developer Options blocked. That is the starting point for the rest of this guide.

Part 3: What we are going to change

MobiControl gets in through three pieces:

- `SunmiInstructions` (system_ext priv-app) enrolls SOTI as Device Owner on first boot.
- `MtkSystemUI` launches SOTI's splash screen on every boot and crashes if SOTI is missing, unless the property `ro.sunmi.welcome=1` is set.
- `GoogleMobiControl` (vendor) is the SOTI agent. `MDMConfig` and `MDMLSample` are helper APKs.

We edit your own backup of `super.bin` offline: delete those APKs, add `ro.sunmi.welcome=1` to `build.prop`, and turn off Android Verified Boot so the modified image is accepted. Do not use a firmware package from the internet for this: even the right model can have a different build, and a mismatched super image will not boot.

Part 4: Disable verified boot (patch vbmeta)

Copy `super.bin`, `boot_a.bin`, `vbmeta_a.bin`, `vbmeta_system_a.bin` and `vbmeta_vendor_a.bin` from the backup into `D:/sunmi/work`. Save this as `patch_vbmeta.py` in the same folder:

```
#!/usr/bin/env python3
import os, struct
OUT = './vbmeta_patched'
os.makedirs(OUT, exist_ok=True)
for name in ['vbmeta_a.bin', 'vbmeta_system_a.bin', 'vbmeta_vendor_a.bin']:
    if not os.path.exists(name):
        raise SystemExit(name + ': file not found')
    data = bytearray(open(name, 'rb').read())
    if data[:4] != b'AVB0':
        raise SystemExit(name + ': not a vbmeta image')
    old = struct.unpack_from('>I', data, 120)[0]
    struct.pack_into('>I', data, 120, 3) # 3 = hashtree + verification
    disabled
    open(os.path.join(OUT, name), 'wb').write(data)
    print(name, 'flags', old, '->', 3, '(', len(data), 'bytes )')
```

Open Ubuntu and run:

```
cd /mnt/d/sunmi/work
python3 patch_vbmeta.py
```

You should see three lines ending in `flags 0 -> 3`, and a new folder `vbmeta_patched`.

Part 5: Find where the partitions sit inside super.bin

`super.bin` is a container (Android dynamic partitions). The offsets published in other guides are for other builds and will be wrong for yours; on this unit `system_a` started at 539 MiB rather than the 546 MiB another guide gave. Save this as `find_partitions.py`:

```
#!/usr/bin/env python3
import struct, sys
FN = sys.argv[1] if len(sys.argv) > 1 else 'super.bin'
f = open(FN, 'rb')
def rd(off, n):
    f.seek(off)
    return f.read(n)
```

```

if struct.unpack_from('<I', rd(4096, 4), 0)[0] != 0x616C4467:
    sys.exit('LP geometry not found: is this a raw super image?')
H = 12288
hdr = rd(H, 256)
if struct.unpack_from('<I', hdr, 0)[0] != 0x414C5030:
    sys.exit('LP metadata header not found')
header_size = struct.unpack_from('<I', hdr, 8)[0]
tables_size = struct.unpack_from('<I', hdr, 44)[0]
p_off, p_num, p_sz = struct.unpack_from('<III', hdr, 80)
x_off, x_num, x_sz = struct.unpack_from('<III', hdr, 92)
tables = rd(H + header_size, tables_size)

def fs_type(off):
    sb = rd(off + 1024, 100)
    if len(sb) < 100:
        return '?'
    if struct.unpack_from('<H', sb, 56)[0] == 0xEF53:
        blocks = struct.unpack_from('<I', sb, 4)[0]
        bsize = 1024 << struct.unpack_from('<I', sb, 24)[0]
        return 'ext4 (%d MiB)' % (blocks * bsize // 1048576)
    if struct.unpack_from('<I', sb, 0)[0] == 0xE0F5E1E2:
        return 'EROFS (cannot be edited with debugfs)'
    return 'unknown'

for i in range(p_num):
    e = tables[p_off + i * p_sz: p_off + (i + 1) * p_sz]
    name = e[:36].split(bytes(1))[0].decode()
    first, num = struct.unpack_from('<II', e, 40)
    if num == 0:
        continue
    exts = []
    for j in range(first, first + num):
        xe = tables[x_off + j * x_sz: x_off + (j + 1) * x_sz]
        sectors, ttype, tdata, tsrc = struct.unpack_from('<QIQI', xe, 0)
        exts.append((tdata * 512, sectors * 512))
    print('')
    print(name, '-', fs_type(exts[0][0]))
    for off, size in exts:
        print('  offset=%d  size=%d' % (off, size))

```

```
python3 find_partitions.py super.bin
```

Write down the `offset` and `size` of `system_a` and `vendor_a`. On the tested unit:

Partition	Offset	Size	Filesystem
vendor_a	174063616	390860800	ext4
system_a	565182464	1468436480	ext4

Your numbers may differ. If a partition shows **EROFS**, this method does not apply to it (EROFS is read-only). `SunmiInstructions` lives in `system_a` (under `system_ext/priv-app`), so there is no separate `system_ext` partition to find.

Check your offsets before editing anything

Substitute your own offset and size:

```
mkdir -p ~/work
dd if=super.bin of=$HOME/work/system_a.img bs=1M iflag=skip_bytes,count_bytes
skip=565182464 count=1468436480 status=progress
debugfs -R 'ls /' ~/work/system_a.img | cat
```

You should see normal Android folders (`system`, `system_ext`, `vendor`, `lost+found`).

`Bad magic number` means the offset is wrong: stop and recheck. Then confirm the files we plan to delete exist:

```
debugfs -R 'ls /system_ext/priv-app' ~/work/system_a.img | cat
debugfs -R 'ls /system/app/MDMConfig' ~/work/system_a.img | cat
debugfs -R 'ls /system/app/MDMLSample' ~/work/system_a.img | cat
```

You should see a folder starting with `SunmiInstructions`, and `MDMConfig.apk` and `MDMLSample.apk`.

Part 6: Edit the images and rebuild super.bin

Save this as `build_super.sh` in `D:/sunmi/work`, change the four numbers at the top to your own, and run `bash build_super.sh`. It only works on copies and stops at the first problem. Unlike the original community script, it keeps the SELinux label and permissions on `build.prop` (losing the label can stop the system from booting), and it verifies each deletion instead of hiding errors.

```
#!/bin/bash
set -e
cd /mnt/d/sunmi/work
SUPER=super.bin
OUT=output/super.bin
W=$HOME/work
mkdir -p output $W

# EDIT: values printed by find_partitions.py
SYS_OFF=565182464
SYS_SIZE=1468436480
VEN_OFF=174063616
VEN_SIZE=390860800
SYS=$W/system_a.img
VEN=$W/vendor_a.img

echo '== 1. Extract partitions =='
dd if=$SUPER of=$SYS bs=1M iflag=skip_bytes,count_bytes skip=$SYS_OFF
count=$SYS_SIZE status=none
dd if=$SUPER of=$VEN bs=1M iflag=skip_bytes,count_bytes skip=$VEN_OFF
count=$VEN_SIZE status=none

echo '== 2. Delete MobiControl APKs =='
SYSDIR=$(debugfs -R 'ls /system_ext/priv-app' $SYS 2>/dev/null | grep -o
'SunmiInstructions[A-Za-z0-9_.+-]*' | head -1)
[ -n "$SYSDIR" ] || { echo 'SunmiInstructions not found'; exit 1; }
APK=$(debugfs -R "ls /system_ext/priv-app/$SYSDIR" $SYS 2>/dev/null | grep -o
'[A-Za-z0-9_.+-]*[.]apk' | head -1)
[ -n "$APK" ] || { echo 'SunmiInstructions APK not found'; exit 1; }
del() {
    echo "Deleting $2"
    debugfs -w -R "rm $2" $1 2>&1 | grep -v '^debugfs' || true
    if debugfs -R "stat $2" $1 2>&1 | grep -q 'Inode: '; then echo "ERROR: $2
still present"; exit 1; fi
}
del $SYS /system_ext/priv-app/$SYSDIR/$APK
del $SYS /system/app/MDMConfig/MDMConfig.apk
del $SYS /system/app/MDMLSample/MDMLSample.apk
del $VEN /operator/app/GoogleMobiControl/GoogleMobiControl.apk

echo '== 3. Patch build.prop =='
debugfs -R "dump /system/build.prop $W/build.prop" $SYS 2>&1 | grep -v
'^debugfs' || true
```

```

[ -s $W/build.prop ] || { echo 'cannot read build.prop'; exit 1; }
[ -z "$(tail -c1 $W/build.prop)" ] || echo >> $W/build.prop
set_prop() {
    if grep -q "^$1=" $W/build.prop; then sed -i "s|^$1=.*|$1=$2|"
$W/build.prop; else echo "$1=$2" >> $W/build.prop; fi
}
set_prop ro.sunmi.welcome 1
set_prop ro.adb.secure 0
set_prop ro.debuggable 1
set_prop persist.sys.usb.config mtp,adb
set_prop persist.service.adb.enable 1
printf 'u:object_r:system_file:s0\0' > $W/ctx
cat > $W/cmds.txt <<EOT
cd /system
rm build.prop
write $W/build.prop build.prop
set_inode_field build.prop mode 0100600
ea_set -f $W/ctx build.prop security.selinux
EOT
debugfs -w -f $W/cmds.txt $SYS 2>&1 | grep -v '^debugfs' || true

echo '== 4. Verify =='
debugfs -R 'stat /system/build.prop' $SYS 2>/dev/null | grep -E
'Mode|selinux'
debugfs -R 'cat /system/build.prop' $SYS 2>/dev/null | grep -E
'ro.sunmi.welcome|ro.adb.secure|ro.debuggable|persist.sys.usb.config|persist.
service.adb.enable'
e2fsck -fn $SYS >/dev/null 2>&1 && echo 'system_a fsck OK' || echo 'system_a
fsck reported problems'
e2fsck -fn $VEN >/dev/null 2>&1 && echo 'vendor_a fsck OK' || echo 'vendor_a
fsck reported problems'

echo '== 5. Rebuild super.bin =='
cp $SUPER $OUT
dd if=$SYS of=$OUT bs=1M oflag=seek_bytes conv=notrunc seek=$SYS_OFF
status=none
dd if=$VEN of=$OUT bs=1M oflag=seek_bytes conv=notrunc seek=$VEN_OFF
status=none
ls -l $SUPER $OUT
cmp -s <(dd if=$OUT bs=1M iflag=skip_bytes,count_bytes skip=$SYS_OFF
count=$SYS_SIZE status=none) $SYS && echo 'system_a OK' || echo 'system_a

```

```
MISMATCH'
```

```
cmp -s <(dd if=$OUT bs=1M iflag=skip_bytes,count_bytes skip=$VEN_OFF
count=$VEN_SIZE status=none) $VEN && echo 'vendor_a OK' || echo 'vendor_a
MISMATCH'
```

What the `build.prop` lines do:

- `ro.sunmi.welcome=1` stops SystemUI from launching the missing SOTI splash screen (without it SystemUI crashes in a loop).
- The other four lines enable ADB without an on-screen authorization prompt.

Good output shows four `Deleting` lines, no `ERROR`, the SELinux label

`u:object_r:system_file:s0` on `build.prop`, all five properties, `output/super.bin` with exactly the same size as `super.bin`, and `system_a OK / vendor_a OK`.

Part 7: Flash

Put the tablet in BROM mode before each command and keep the cable connected while `super` is written (10 to 20 minutes).

```
python mtk.py w vbmeta_a D:/sunmi/work/vbmeta_patched/vbmeta_a.bin
python mtk.py w vbmeta_system_a
D:/sunmi/work/vbmeta_patched/vbmeta_system_a.bin
python mtk.py w vbmeta_vendor_a
D:/sunmi/work/vbmeta_patched/vbmeta_vendor_a.bin
python mtk.py w super D:/sunmi/work/output/super.bin
python mtk.py e metadata
```

If any command fails, stop and fix it before running the next one. You do not need to touch `boot_a` yet.

To undo everything at this point, write the original `super.bin` and the three original vbmeta files from your backup with the same commands, then `python mtk.py e metadata`.

Part 8: Fix the boot screen and wipe data

"Red state" boot warning

If the boot screen says **Red state** and the tablet reboots in a loop, the bootloader has locked itself again. Unlock it once more:

```
python mtk.py da seccfg unlock
```

The warning should now read **Orange state**, which is normal for an unlocked bootloader.

Wipe userdata through recovery

After the super change, the old userdata (encrypted, and holding MobiControl's enrollment) must be wiped, otherwise the tablet keeps rebooting. First try the stock recovery: power off, hold Vol Up + Power (or Vol Down + Power), and choose **Wipe data/factory reset** with Vol Down to move and Power to select.

If you cannot reach recovery, tell it to wipe on next boot. This unit has no `misc` partition; the equivalent is `para` (check with `python mtk.py printgpt`). Read it first:

```
python mtk.py r para D:/sunmi/work/para_old.bin
```

Keep `para_old.bin` as a backup. Save this as `make_para.py` and run `python3 make_para.py` in Ubuntu (the `assert` checks that the file is the 524288-byte size of this unit's `para`; adjust if yours differs):

```
d = bytearray(open('para_old.bin', 'rb').read())
assert len(d) == 524288, 'unexpected size'
d[0:32] = b'boot-recovery'.ljust(32, bytes(1))
msg = b'recovery\n--wipe_data\n--locale=en_US\n'
d[64:64 + len(msg)] = msg
open('para_recovery.bin', 'wb').write(d)
```

Write it back and boot:

```
python mtk.py w para D:/sunmi/work/para_recovery.bin
```

Unplug and power on. The tablet wipes itself and restarts. The first boot takes 3 to 5 minutes and the boot animation loops; that is normal.

Check the result

You should now have a normal home screen, no SOTI prompt, and a working **Developer options** page. With the tablet connected by USB (and USB debugging allowed):

```
./adb devices
./adb shell dpm list-owners
```

The tablet should be listed, and `list-owners` should print nothing (no Device Owner).

Part 9: Root with Magisk

Do **not** use `adb root`: on this firmware the USB connection drops and does not come back. Patch the boot image with the Magisk app instead.

1. Download the latest `Magisk-vXX.X.apk` from <https://github.com/topjohnwu/Magisk/releases>, rename it `magisk.apk` and put it in `C:/platform-tools`.
2. Install it and put your unmodified `boot_a.bin` (from the backup) on the tablet:

```
./adb push D:/sunmi/work/boot_a.bin /sdcard/Download/boot.img
./adb install magisk.apk
```

3. On the tablet open **Magisk > Install > Next > Select and Patch a File**, pick `Download/boot.img`, leave **Patch vbmeta in boot image** switched off, tap **Let's Go**, and wait for **All done!**.
4. Copy the patched file back (use the exact name Magisk shows):

```
./adb pull /sdcard/Download/magisk_patched-XXXXX.img
D:/sunmi/work/magisk_boot.img
```

It must be smaller than 32 MB, the size of the `boot_a` partition.

5. Flash it (BROM mode):

```
python mtk.py w boot_a D:/sunmi/work/magisk_boot.img
```

6. Boot the tablet, open Magisk, accept the additional setup if asked, and let it reboot. Then test:

```
./adb shell su -c id
```

Tap **Grant** on the tablet. The output should start with `uid=0(root)`.

Undo: write the original `boot_a.bin` back with `python mtk.py w boot_a`.

Part 10: Add Google Play

Why a plain APK install fails

If you sideload Google Play Services, Play Store and Services Framework as ordinary APKs, Play Store crashes at start-up with `SecurityException: You either need MANAGE_USERS or CREATE_USERS permission`. Those permissions are only granted to privileged system apps. The tablet's system partition is nearly full, and `ro.control_privapp_permissions` is set to `enforce` (a privileged app without a matching permission file can prevent boot), so we use a Magisk module: it overlays the Google files onto the system at boot without writing to the system partitions.

10.1 Build the module

Download **MindTheGapps-11.0.0-arm64-20230922_081122.zip** from <https://github.com/MindTheGapps/11.0.0-arm64> (Releases). Use the plain `arm64` file, not `arm` and not `ATV`. Put it in `C:/platform-tools`. In Ubuntu:

```
cd /mnt/c/platform-tools
unzip -q MindTheGapps-11.0.0-arm64-20230922_081122.zip -d gapps
M=module
mkdir -p $M/system/product/priv-app $M/system/product/etc
$M/system/system_ext/priv-app $M/system/system_ext/etc
cp -r gapps/system/product/priv-app/PrebuiltGmsCore
gapps/system/product/priv-app/Phonesky $M/system/product/priv-app/
cp -r gapps/system/system_ext/priv-app/GoogleServicesFramework
$M/system/system_ext/priv-app/
cp -r gapps/system/product/etc/permissions gapps/system/product/etc/sysconfig
$M/system/product/etc/
cp -r gapps/system/system_ext/etc/permissions $M/system/system_ext/etc/
printf 'id=mtg_gapps\nname=MindTheGapps
minimal\nversion=1\nversionCode=1\nauthor=local\ndescription=Play Services,
Play Store, GSF\n' > $M/module.prop
```

This keeps only the three core apps (Play Services, Play Store, Services Framework) plus every permission and sysconfig file. Leaving out the setup wizard, voice search and other extras keeps the risk low.

10.2 Install the module

In PowerShell:

```
cd C:/platform-tools
./adb push module/. /data/local/tmp/mtg_module
./adb shell
```

Then, inside the tablet's shell, type these lines one at a time:

```
su
mkdir -p /data/adb/modules/mtg_gapps
cp -a /data/local/tmp/mtg_module/. /data/adb/modules/mtg_gapps/
chown -R root:root /data/adb/modules/mtg_gapps
find /data/adb/modules/mtg_gapps -type d -exec chmod 755 {} +
find /data/adb/modules/mtg_gapps -type f -exec chmod 644 {} +
chcon -R u:object_r:system_file:s0 /data/adb/modules/mtg_gapps/system
ls /data/adb/modules/mtg_gapps/system/product/priv-app
exit
exit
```

The `ls` must show `PrebuiltGmsCore` and `Phonesky`. Accept the Magisk superuser prompt on the tablet when it appears.

If you had already sideloaded any Google APKs, remove them first, otherwise they conflict with the module:

```
./adb shell pm uninstall com.android.vending
./adb shell pm uninstall com.google.android.gms
./adb shell pm uninstall com.google.android.gsf
./adb shell pm uninstall com.google.android.gsf.login
```

10.3 Reboot and verify

```
./adb reboot
```

The first boot can take up to 6 minutes. Then:

```
./adb shell pm list packages -f | Select-String 'gms|vending|gsf'
```

The paths should start with `/product/priv-app/` or `/system_ext/priv-app/`.

If something goes wrong: to switch the module off, run `./adb shell su -c 'touch /data/adb/modules/mtg_gapps/disable'` and reboot. If the tablet will not boot at all, write the original `boot_a.bin` back (Part 9, undo): Magisk and the module are then not loaded and nothing on the system partitions has changed.

Part 11: Sign in and install DoorDash

1. Open **Play Store** and sign in with a Google account (consider a dedicated account for this device).
2. If Play Store says the device is not certified, or will not download apps, register it: install **Device ID** from Play Store and copy the **Google Service Framework (GSF)** ID, then open <https://www.google.com/android/uncertified> on a PC, sign in with the same account, paste the ID and register. Wait 10 to 30 minutes, then clear storage for **Play Store** and **Play Services** (not Services Framework, which would generate a new ID) and reboot.
3. Install **DoorDash Order Manager** (or a browser) and sign in with the merchant account.
4. Give the app notification permission, set its battery usage to **Unrestricted**, and keep the tablet on charge.

Troubleshooting

Symptom	Likely cause and fix
<code>Bad magic number</code> in debugfs	Wrong offset for that partition; rerun <code>find_partitions.py</code> and recheck
<code>LP geometry not found</code>	<code>super.bin</code> is not a raw image, or the backup is damaged
Boot screen says Red state	Bootloader locked again: <code>python mtk.py da seccfg unlock</code>
Bootloop after flashing super	Userdata not wiped; do the Part 8 wipe
SystemUI keeps crashing	<code>ro.sunmi.welcome=1</code> missing from <code>build.prop</code>
<code>adb root</code> makes the device disappear	Do not use it; use <code>su</code> through Magisk
Play Store crashes with <code>MANAGE_USERS</code>	Google apps installed as ordinary APKs; use the Part 10 module
<code>adb: no devices</code>	Use a data cable, set USB mode to File transfer, turn USB debugging off and on

Limits and cautions

- **Printing:** DoorDash Order Manager supports specific printer models. I have not verified that it can use the Sunmi built-in receipt printer, so confirm printer compatibility with DoorDash merchant support before relying on this setup for receipts.
- **Not certified:** the device is rooted and uncertified, so it fails Play Integrity checks. Banking apps, and any app that enforces device integrity, may refuse to run.
- **Open ADB:** the `build.prop` edits make ADB accessible without an authorization prompt. If the tablet will sit on a counter or a shared network, consider reversing `ro.adb.secure` and `ro.debuggable` (edit `build.prop` offline the same way and reflash).
- **Updates:** do not apply a vendor OTA update; it would overwrite your changes.

Credits

Built on mtkclient (bkerler), Magisk (topjohnwu), MindTheGapps, e2fsprogs `debugfs`, and a community guide for the Just Eat variant of this tablet.