

# Dissecting the Modern Android Data Encryption Scheme

---

Maxime Rossi Bellom  
Damiano Melotti



Quarkslab

# Who we are



- [@DamianoMelotti](#)
- Security researcher @ Quarkslab
- Interested in low-level mobile security and fuzzing

- [@max\\_r\\_b](#)
- Security researcher and R&D leader @ Quarkslab
- Working on mobile and embedded software security



# The trigger

*> Hey! My device fell into water and the main SoC is dead. However the Titan M<sup>1,2</sup> chip seems to be alive and well, do you think you would be able to help me recover my data on the phone?*

[1]: [2021: A Titan M Odyssey](#) (Maxime Rossi Bellom, Philippe Teuwen, Damiano Melotti)

[2]: [Attack on Titan M, Reloaded: Vulnerability Research on a Modern Security Chip](#) (Damiano Melotti, Maxime Rossi Bellom)

# The trigger

- Our answer: no, the main SoC is still essential for disk encryption/decryption
  - ... but up to what extent?
- Objective of this research: find out exactly
- Offensive approach:
  - What would a forensic analyst do?
  - Assuming infinite vulnerabilities, what can you do to get the secrets out?
  - Do you still need to bruteforce credentials?

# Data Encryption at Rest 101

- Idea: no sensitive plaintext files in storage
  - Attackers must not find files in clear on disk
- Threat model: full physical access to powered-off device
- Data is automatically encrypted when written and automatically decrypted when read
- How?
  - Android: Full-Disk Encryption and File-Based Encryption (required from Android 10)
  - Underneath: dm-crypt for FDE, fscrypt for FBE

# File-Based Encryption at Rest 101

- Relies on fscrypt, implemented in the Linux kernel
  - It supports Ext4, F2FS, and UBIFS
- Operates at the filesystem level
  - Allows files encrypted with different keys or unencrypted in a file system
- A master key is provided for directory tree
- Then derived per file keys (for regular file, directory, and symbolic link)
- Metadata are not encrypted by fscrypt

# Android File-Based Encryption

- Each file has its own key
- Direct Boot and multi-user support
- Two encryption levels:
  - Credential Encrypted (CE), available only after authentication
  - Device Encrypted (DE), available also during boot
- In short, 2 “main” keys
  - DE key, for data decrypted at boot
  - CE key, available after authentication, protecting user data
- DE key is automatically decrypted using HW-backed keys

# Android File-Based Encryption

```
a22x:/ # ls /data/data | head -n 20
+BWqxAAAAAQGI050Z57bZxMl6oTCNKC
+geIWCAAAAgIsCJB+mpPqIQYOH0FrnojC1KJ8e0lvGZWJYXWFTc0WD
+1E39AAAAAwNvineHXItKqcE1Glo9+EinbUaa3wvp,fXEjaJ3r4BiC
+ostbBAAA AwZ2g592ei8GG2DMfc4y6H94jxkEfoRzUDdlQZn0YZKqA5VCUgi89sf2JN8yCBFraC
,0wF+BAAA AgQtSCvHd5rRmVC2lVxEHt1Eo50M9kma1oe+vkWT176K8dxofVp5RcmnLv0xC7WMLJ
,WGCADAAA Ag66o3+mZ7f009fBNp8zQSdWBqRJIwPUZHafQiTu7khxB
,eGnGDAAA Aw+ZAptd14KRyth5ncJmJkYZBTBW7DoUNpMamRGj05MA
,vu29CAAAA AgKApb3amvoChi0pYILwr5xxvUtf0TpZ2h6Cr0wG4CR5D
0MQ5GCAAAA QAtXNLN524L3GwuGAek+rVelNjgE5mwqljis0kydZa2FKJlVD6ezJJGkcjcRTTD7B
0QcmpBAAA AwMxohHqP+AT3ktRtIAJk9, bu2g04xwjLzf, vgN71QQ2B
0WZ9mBAAA A5KuJUTYZ61eUrgnVJAloJnZDm3b7JybYYPw8xhLrf0D
11ppkCAAAA I4M8ENP6k0t3rA9kIN9bepZ0FdpYDp6bQ5Cj2IU2NB
13nD7DAAA Aw, NoobtAOXesI0kFqFC4MIwoKYAfdBQrGTm8rfbbdsNC
1K0haDAAA AibCc dZY7LE6W4+DAXnAiv, PEhu4CiUL6ofn9ZuL6buC
1V6luAAA AwmrNINeM84y83, LYzeNEkdCcACNx1AhJ+sI7VzbV51yuebjw9M0PTTrjRi9cv391kD
1fMeDAAA AgPnPYDGu0WWC8uWiojo+nhhnqWk+x0ZerDfrWY3ZVIbB
1hGhtCAAAA AwfxP3KKXNUYQ0SWoBzqZcsbu2g04xwjLzf, vgN71QQ2B
2d3fjDAAA Agf7ilkMftxkJiD0JbRx9, dtg04Rk9L1P5CCl7JuqqewNpAl07TQq0KJh2yXzcj9, 8JfbjIHwnpEmj7FQ0ixXBP
2v, UXBAAA QloHyVK08uify8onfmrQXJawy1Dqwq6kc2g2BKr0H48D
2wjJhBAAA AgQ0MvGF3NqTpssmh6XgAWw1jHW986vSwQMXIPQQz6ghD
```

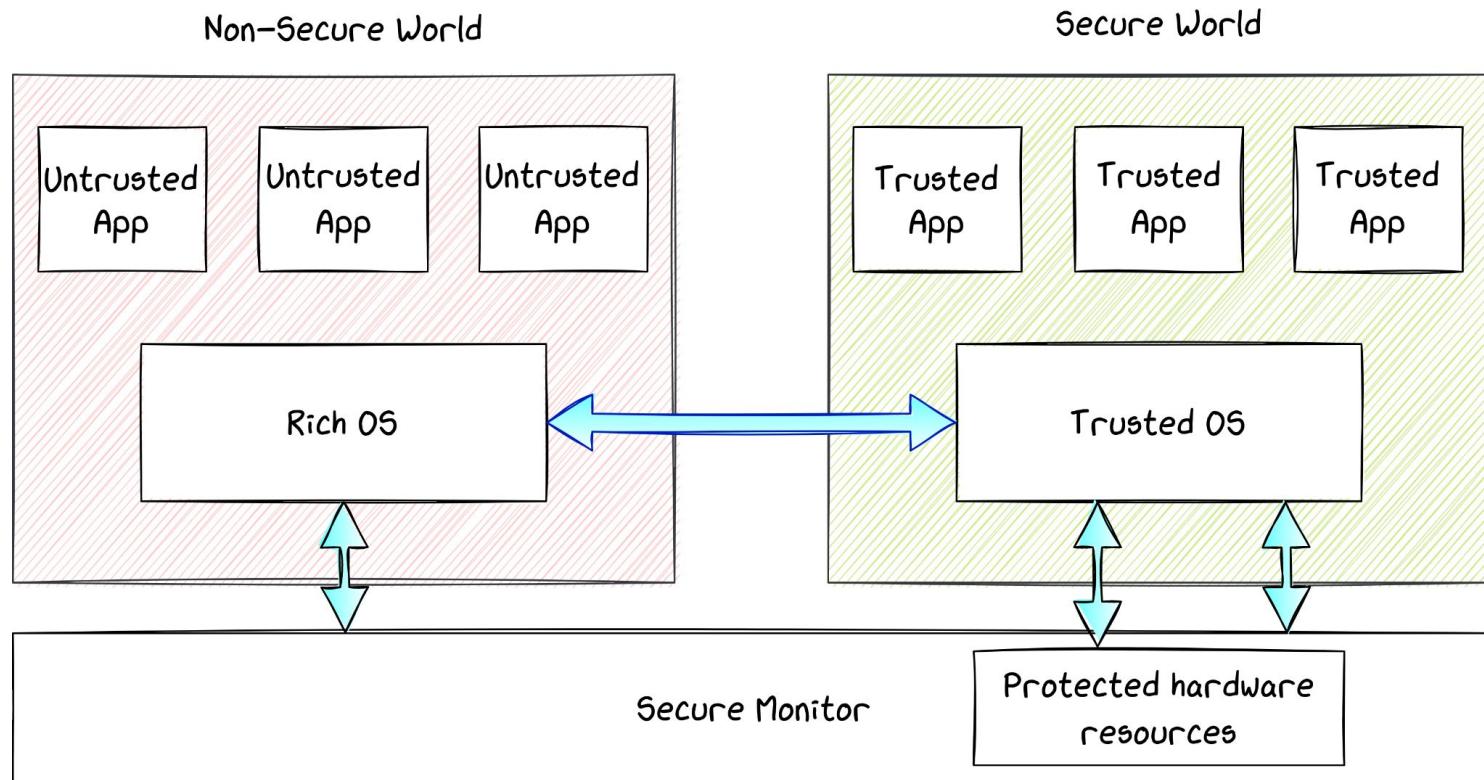
# FBE key derivation

- We focus only on the CE key
- Complex derivation steps
  - Start from DE files owned by privileged users

```
a22x:/ # ls -l /data/system_de/0/spblob/
total 32
-rw----- 1 system system      58 2022-06-29 21:59 0000000000000000.handle
-rw----- 1 system system      72 2022-06-29 21:59 921e9ab09af8d9d.metrics
-rw----- 1 system system     93 2022-06-29 21:59 921e9ab09af8d9d.pwd
-rw----- 1 system system 16384 2022-06-29 21:59 921e9ab09af8d9d.secdis
-rw----- 1 system system     186 2022-06-29 21:59 921e9ab09af8d9d.spblob
```

- User credentials are used in the process
  - No matter how many bugs an attacker has, bruteforcing remains necessary!

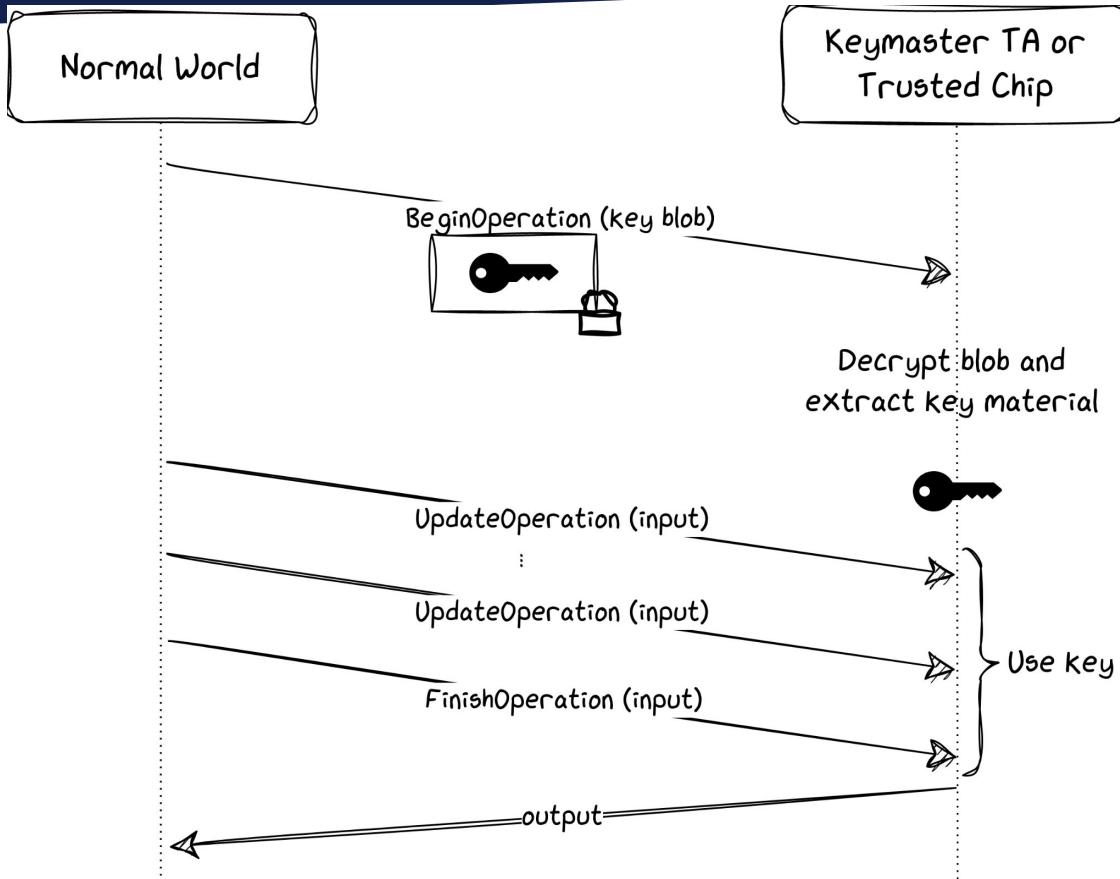
# ARM TrustZone

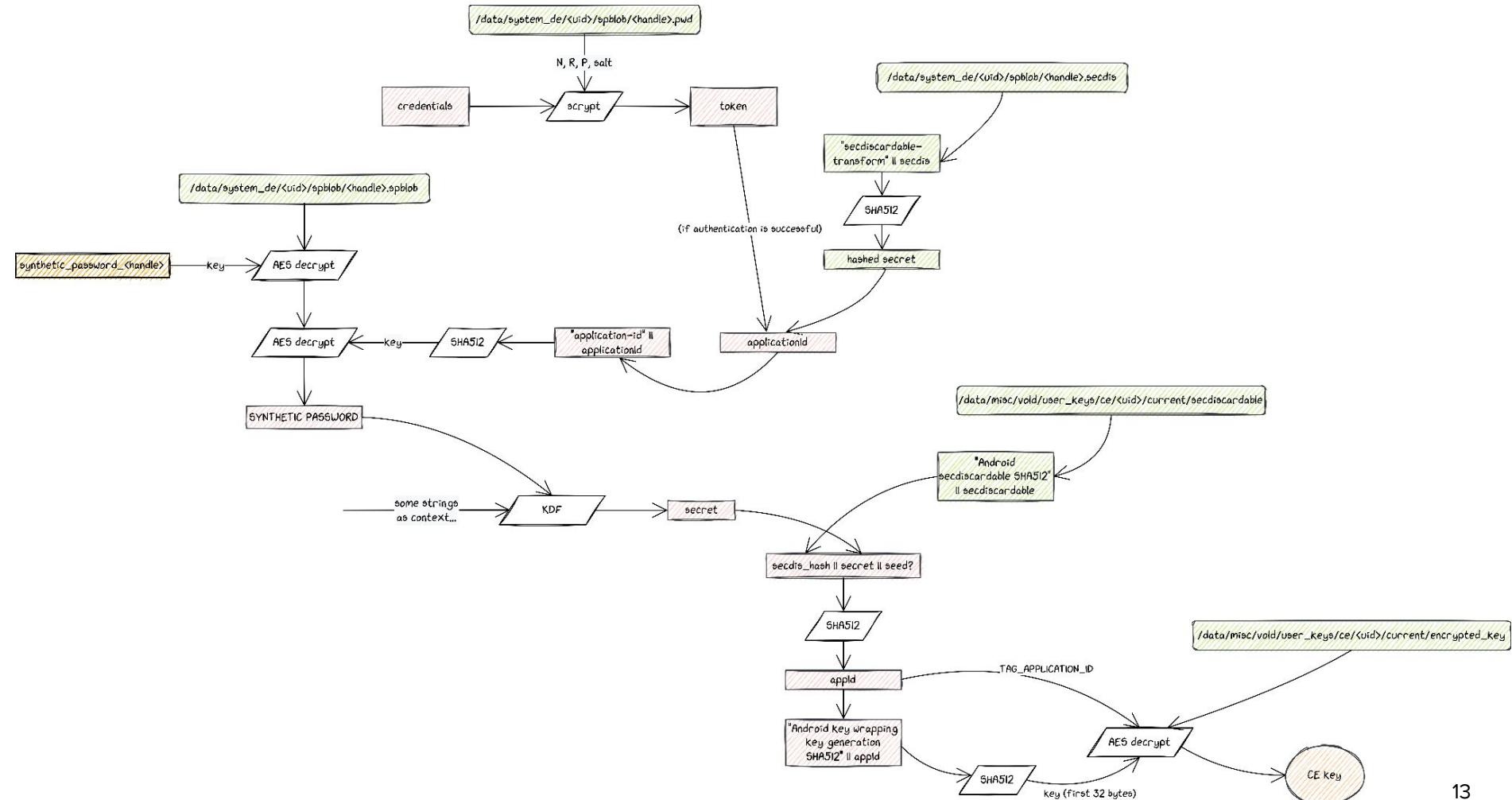


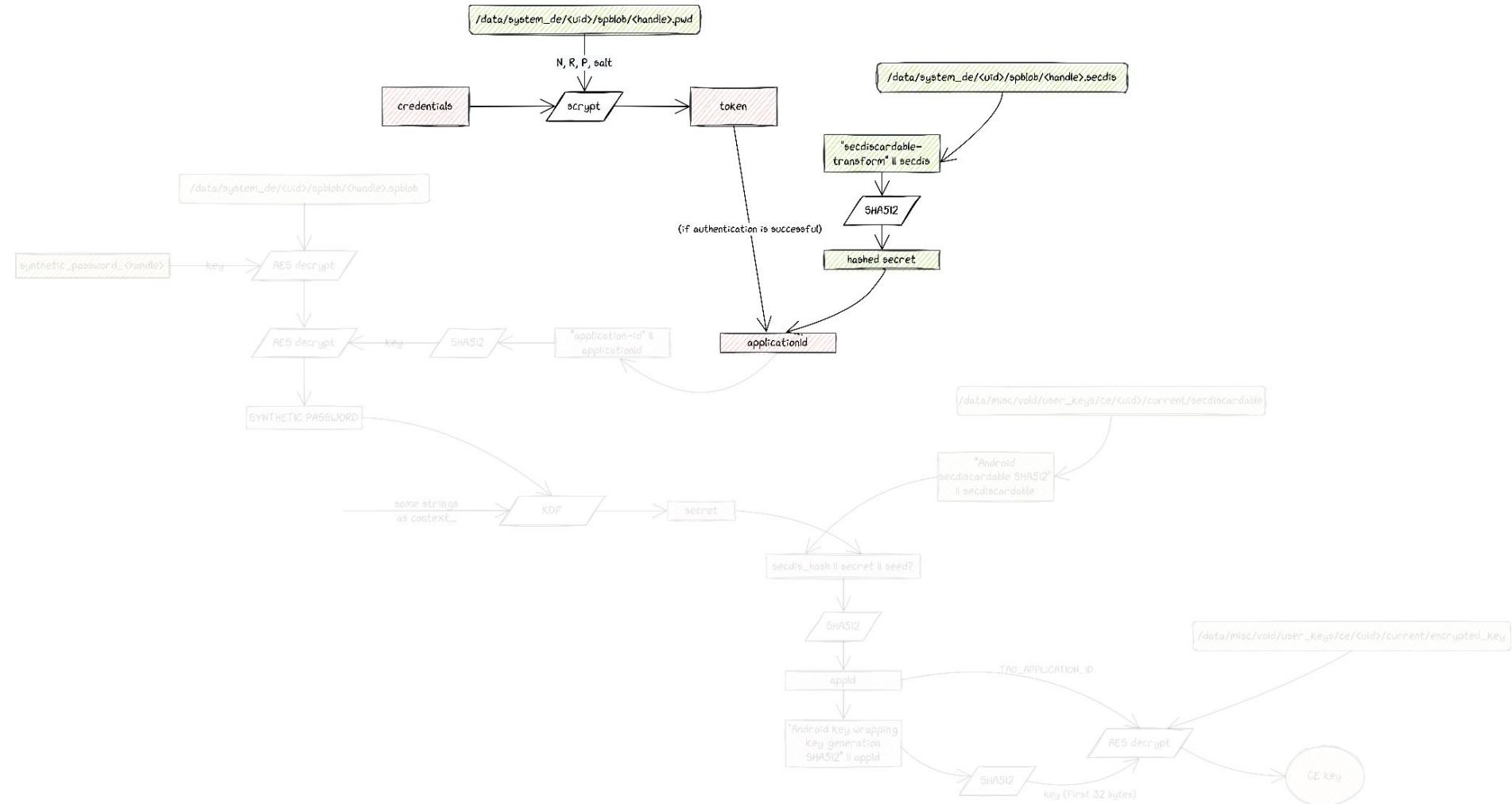
# Android Keystore system

- Key storage and crypto services
- Keys are stored as *key blobs*
- Three protection levels:
  - Software only
  - TEE (default)
  - Hardware-backed (StrongBox)
- Raw key should never leave protected environment

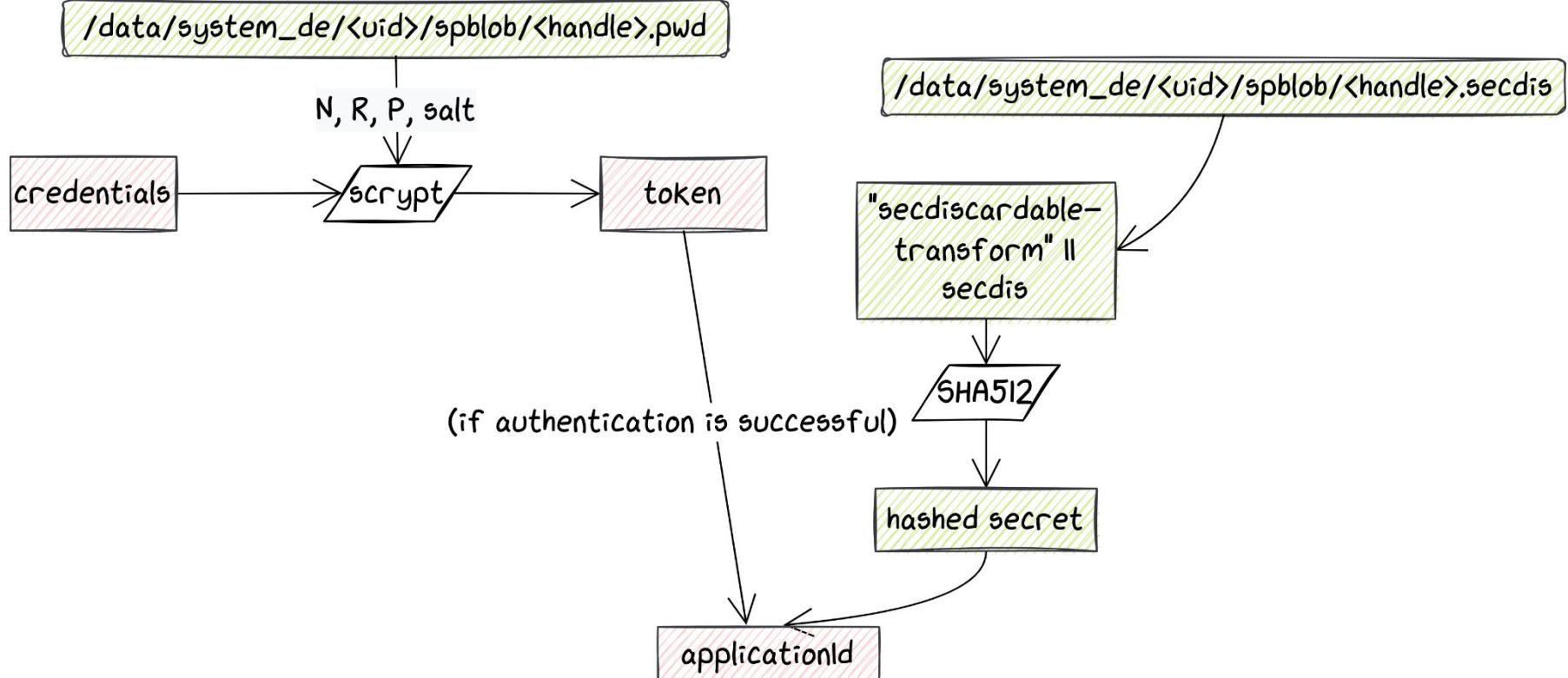
# Android Keystore system







# Credentials, scrypt, secdis

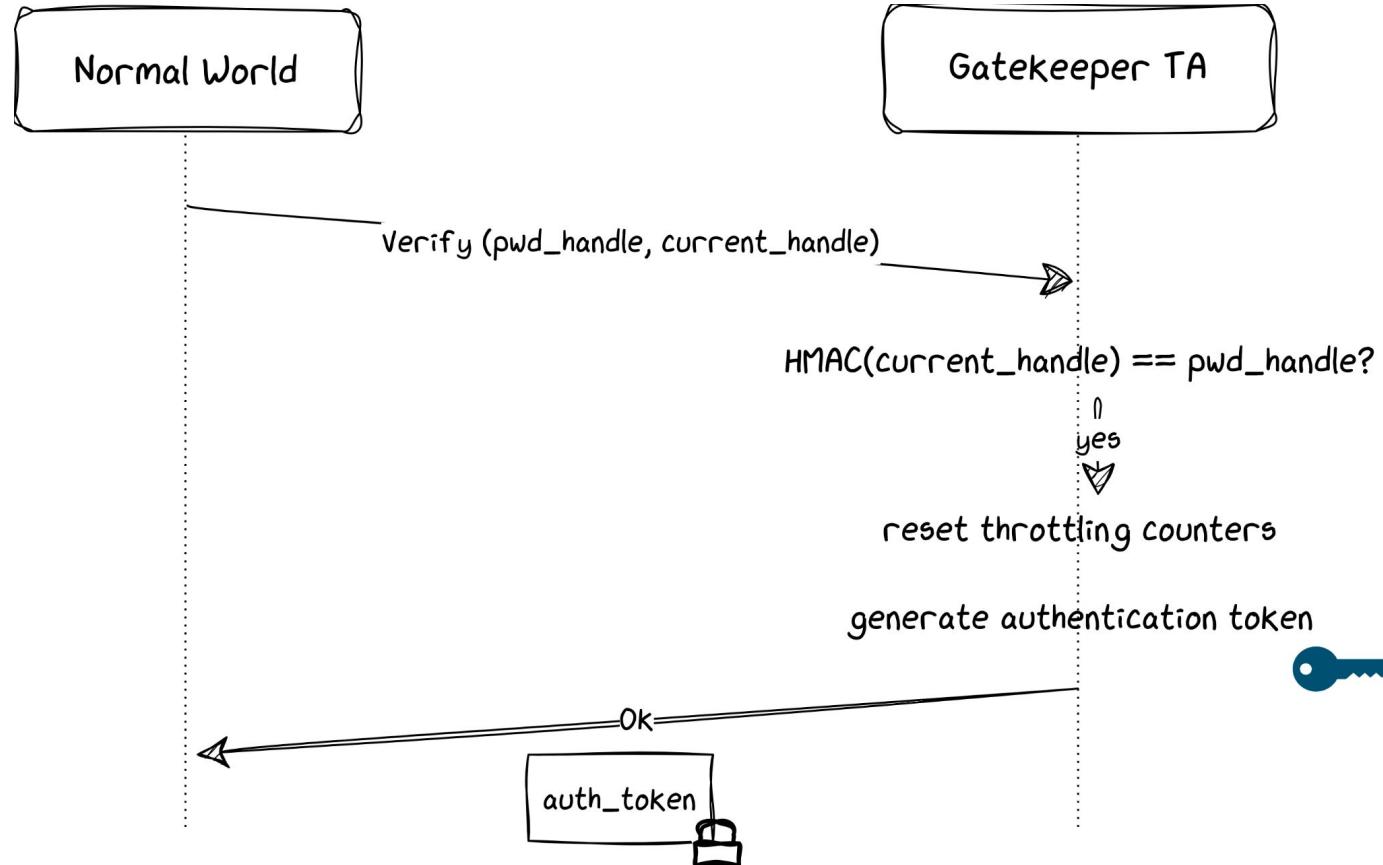


# Authentication with Gatekeeper

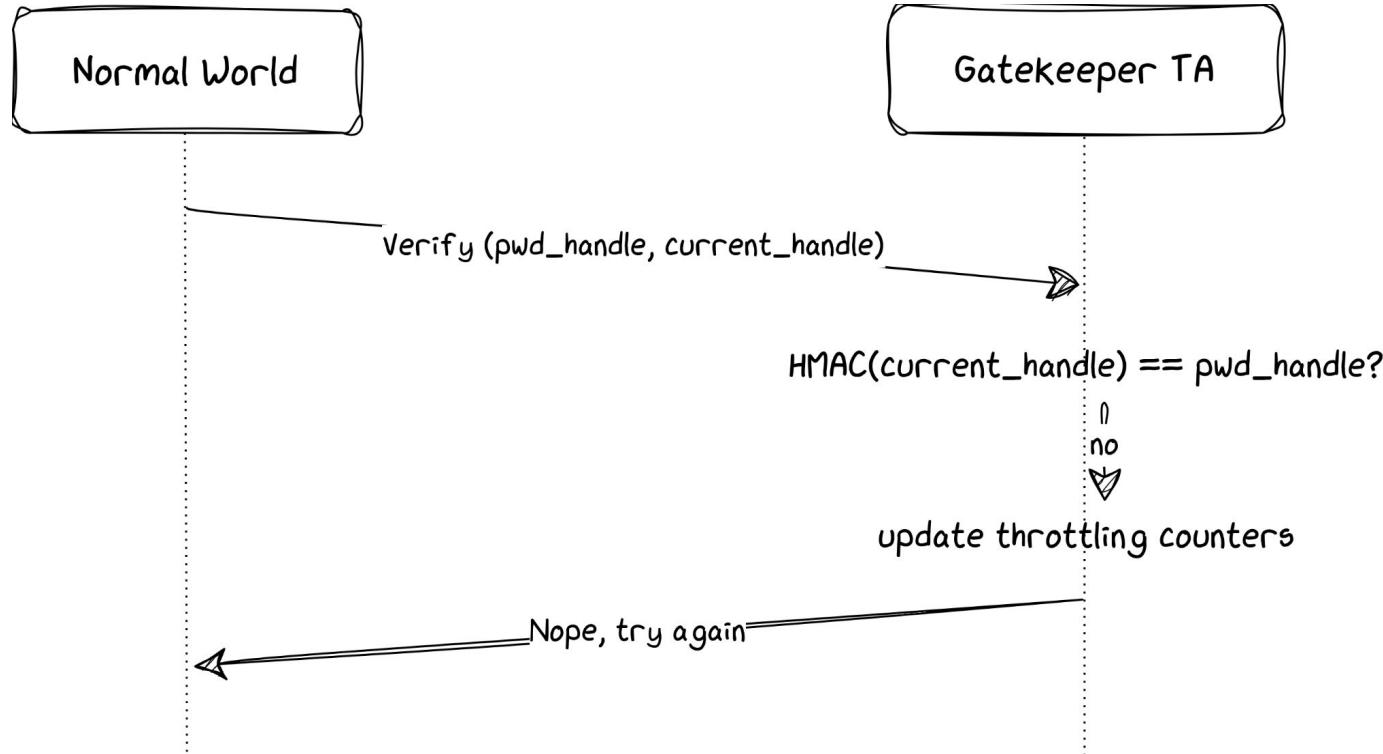
- The Gatekeeper TA verifies credentials from the TEE
- `/data/system_de/<uid>/spblob/<handle>.pwd`
  - scrypt parameters
  - *password handle*, i.e. `HMAC(SHA512("user-gk-authentication" || scrypt(credentials, params))`
- If successful, Gatekeeper returns an *authentication token*
  - Signed token to be used to prove successful authentication
  - Needed by Keymaster to use authentication-bound keys
  - Standard format, designed not to allow replay attacks<sup>3</sup>
- Gatekeeper implements throttling to prevent bruteforcing

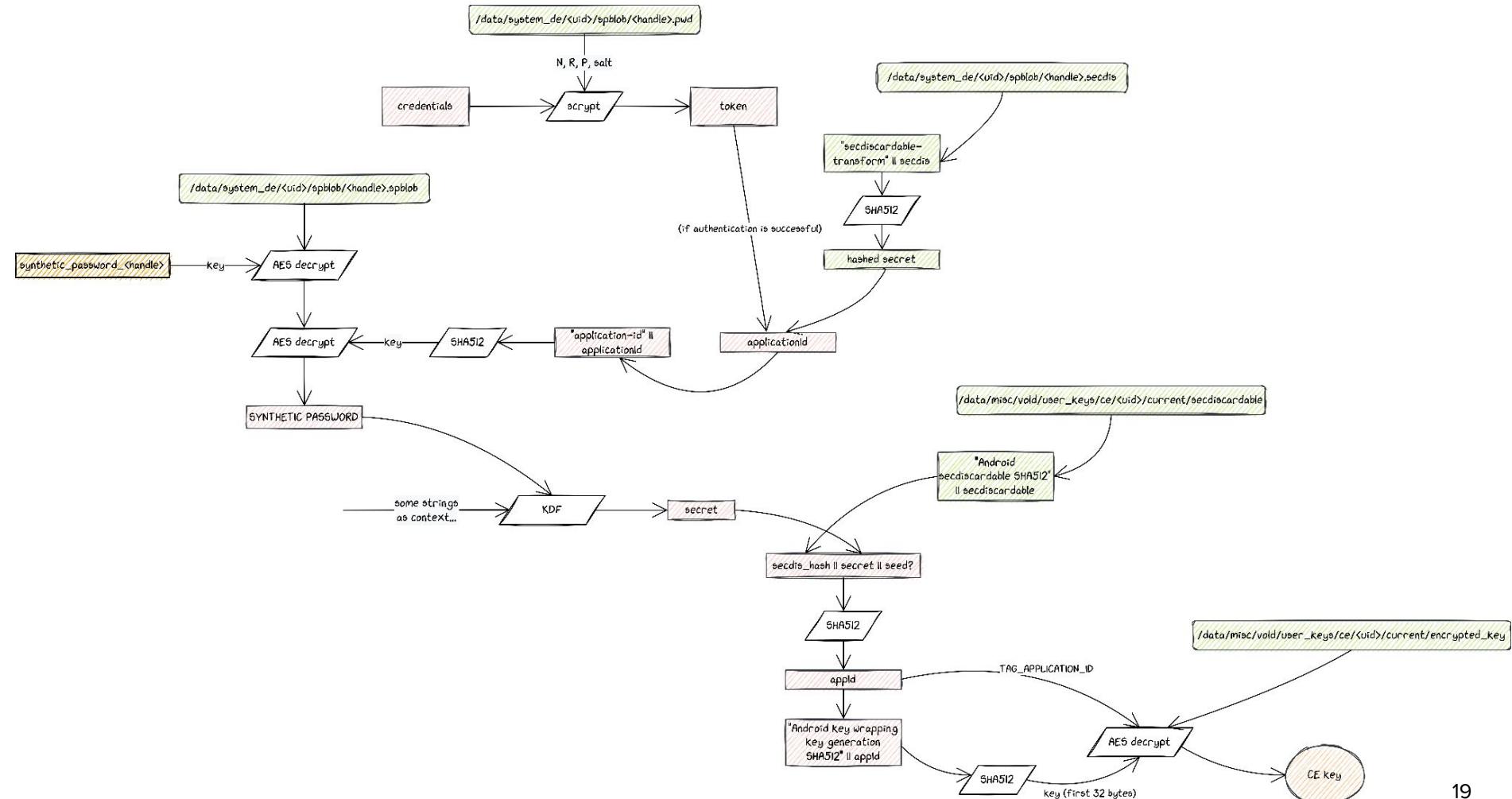
[3]: [https://android.googlesource.com/platform/hardware/libhardware/+/master/include/hardware/hw\\_auth\\_token.h](https://android.googlesource.com/platform/hardware/libhardware/+/master/include/hardware/hw_auth_token.h)

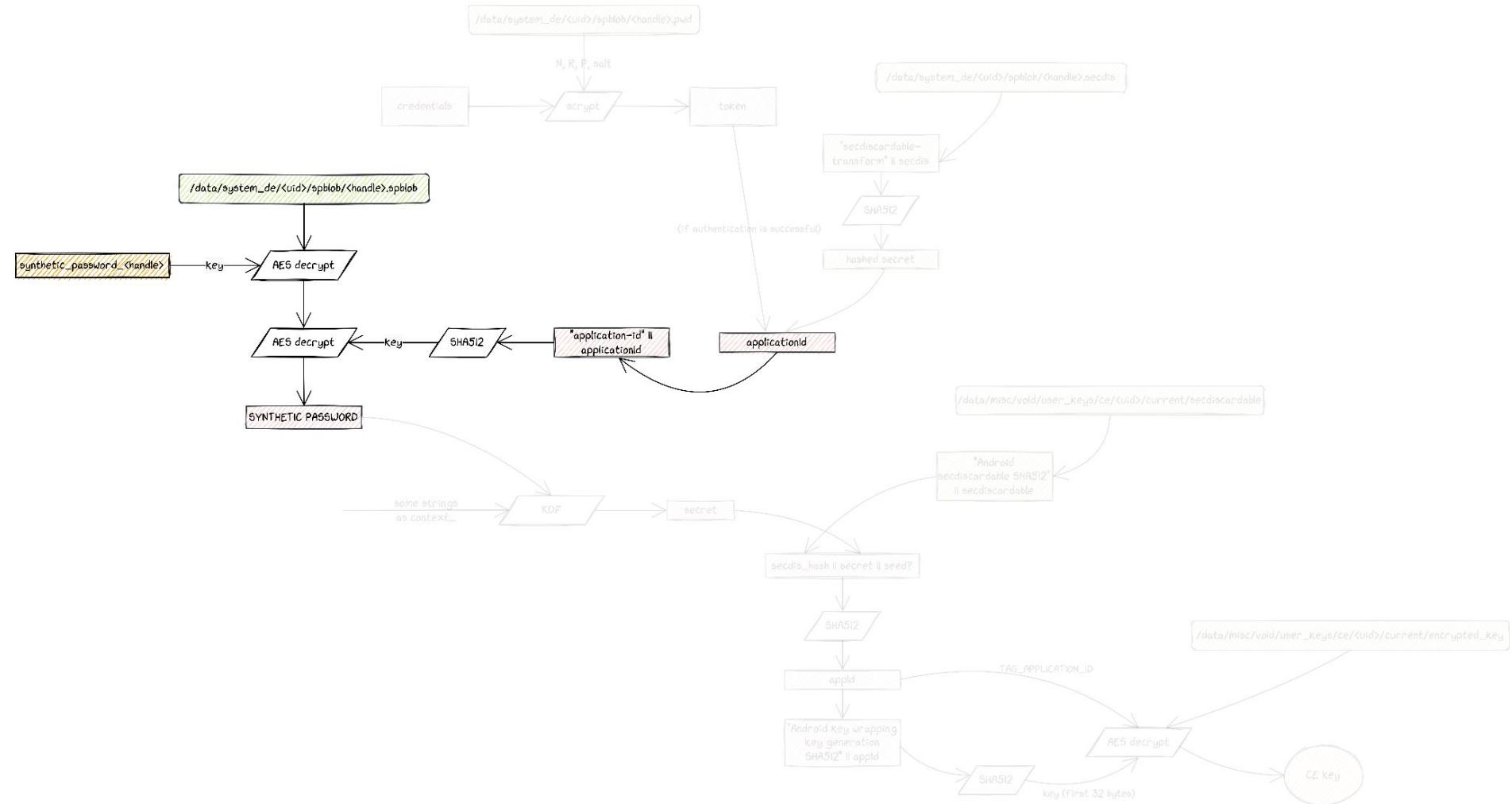
# Successful authentication



# Failed authentication

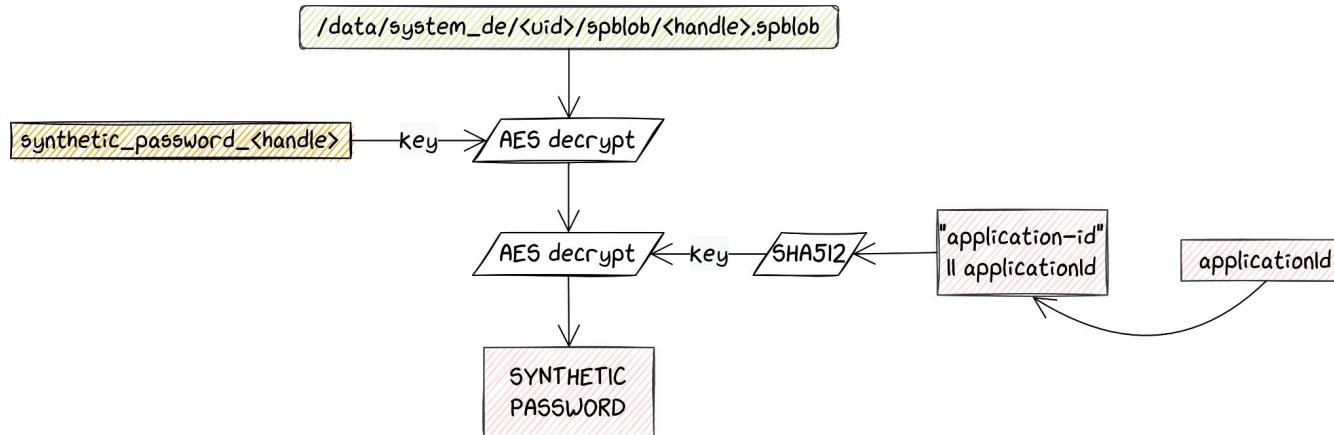






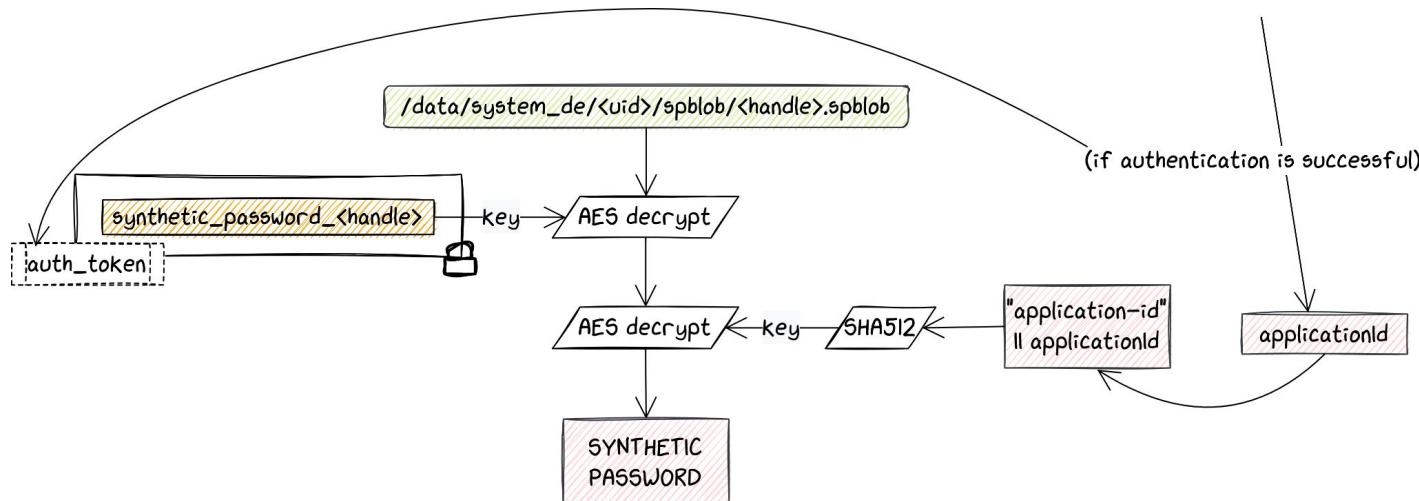
# Synthetic Password

- Problem: credentials shouldn't be linked to the CE key
  - What if the user changes them?
- Solution: Synthetic Password
  - Key blob stored in `/data/system_de/<uid>/spblob/<handle>.spblob`
  - First, decrypted with an authentication-bound, TEE-protected key
  - Then, decrypted with the (hashed) applicationId



# Attacking SP derivation

- Need to target the TEE
- Two alternatives
  - Keymaster TA (accessing the first AES key)
  - **Gatekeeper TA** (validating credentials and minting auth tokens)



# Global strategy

- Our goal
  - Root the device and access all the device encrypted files
  - Patch the Gatekeeper trustlet to accept any credentials
- For that we need
  - Either multiple bugs (code exec, priv esc, etc)
  - Or one critical bug early in the boot process

# PoC on Samsung Device

- Samsung A225f and A226b
  - Cheap (~250€)
  - Mediatek SoC MT6769V and MT6833V
  - No security chip
  - Mix of Mediatek and Samsung code
  - Trustzone OS: TEEGRIS
  - Known critical Boot ROM vulnerability



# The Boot ROM Known Vulnerability

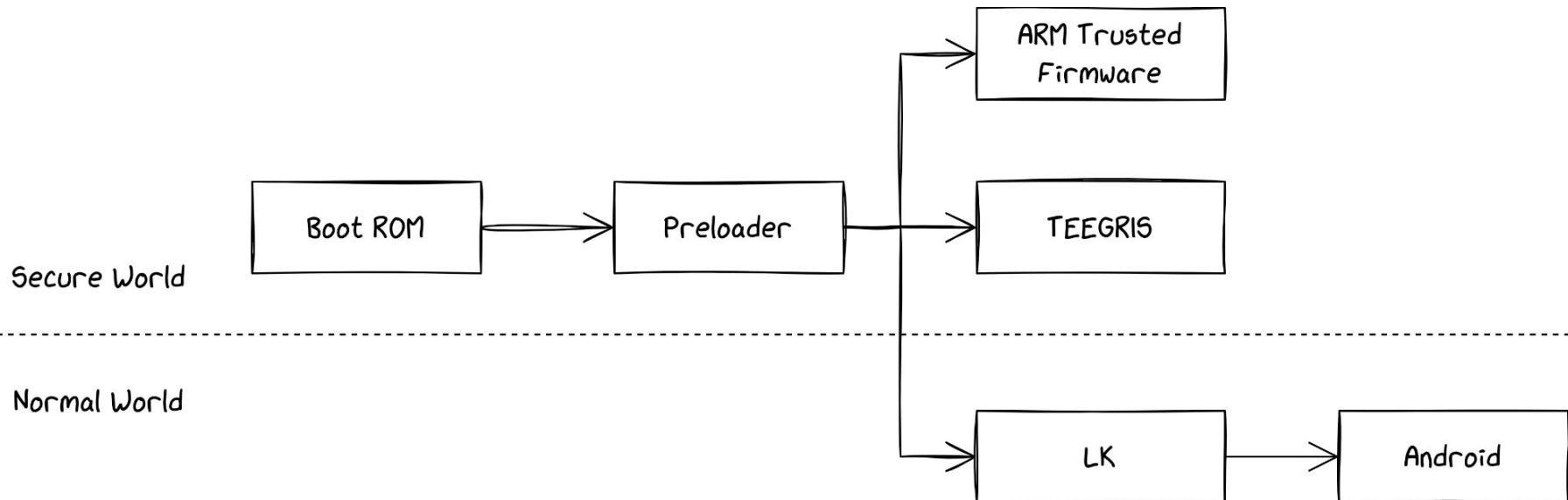
We use the project MTKClient<sup>4</sup> (by Bjoern Kerler – [@viperbjk](https://github.com/bkerler/mtkclient))

- Exploit boot ROM bugs impacting plenty of Mediatek SoC

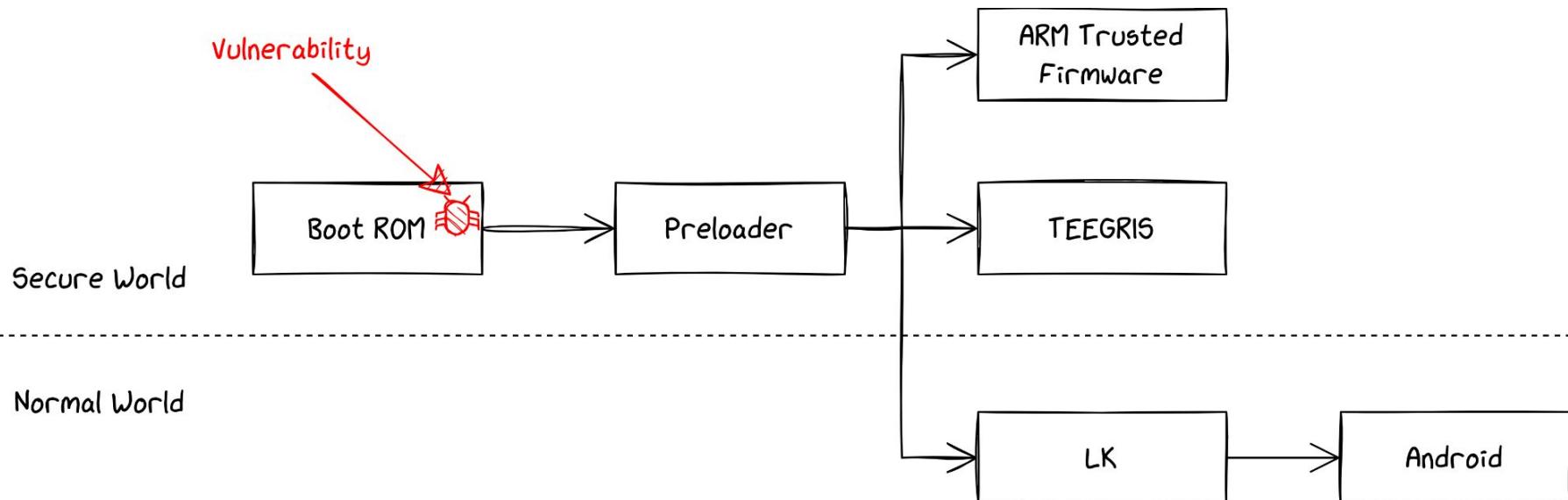
In short, we use it to

- Read/write all the partitions we need to patch
- Boot a patched preloader (BL2) image
- Bypass the secure boot checks done in boot ROM and preloader
- It just works :)

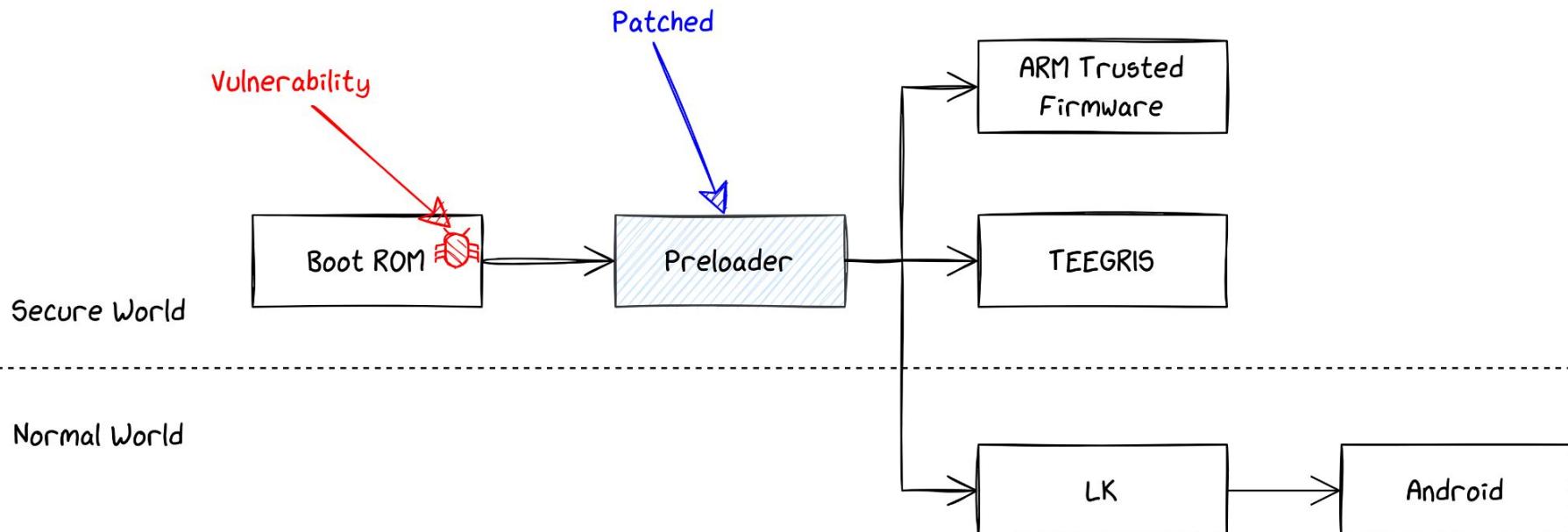
# Boot Process



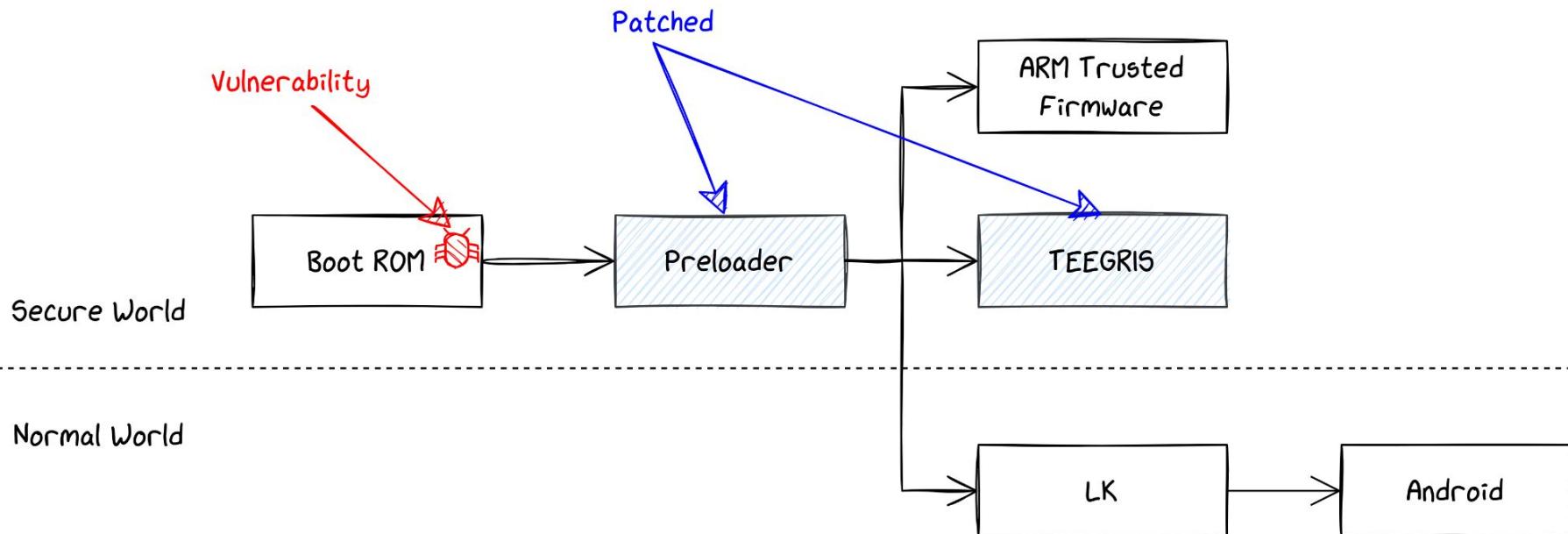
# Boot Process



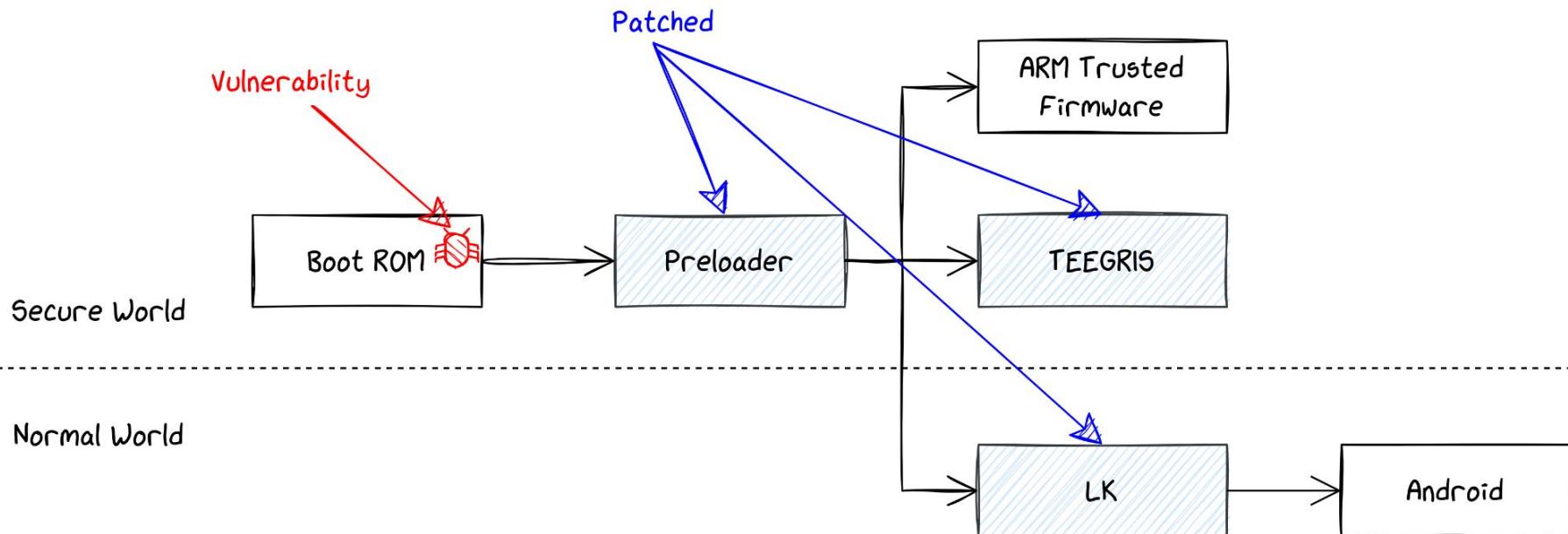
# Boot Process



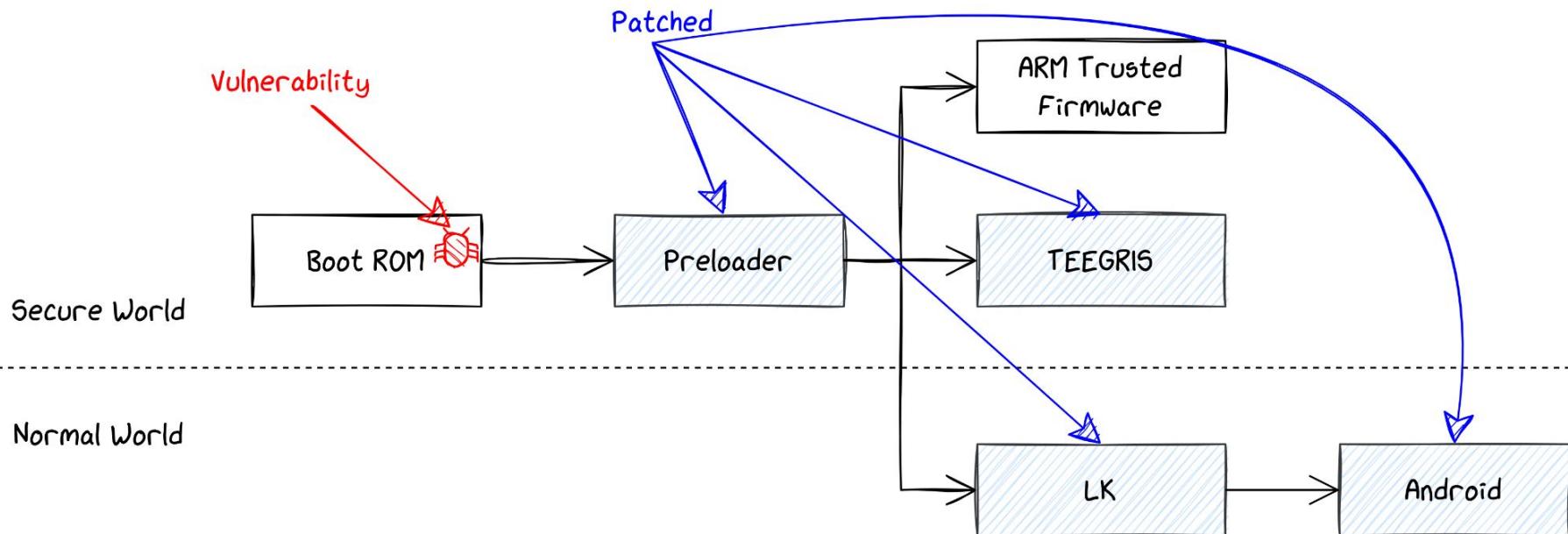
# Boot Process



# Boot Process

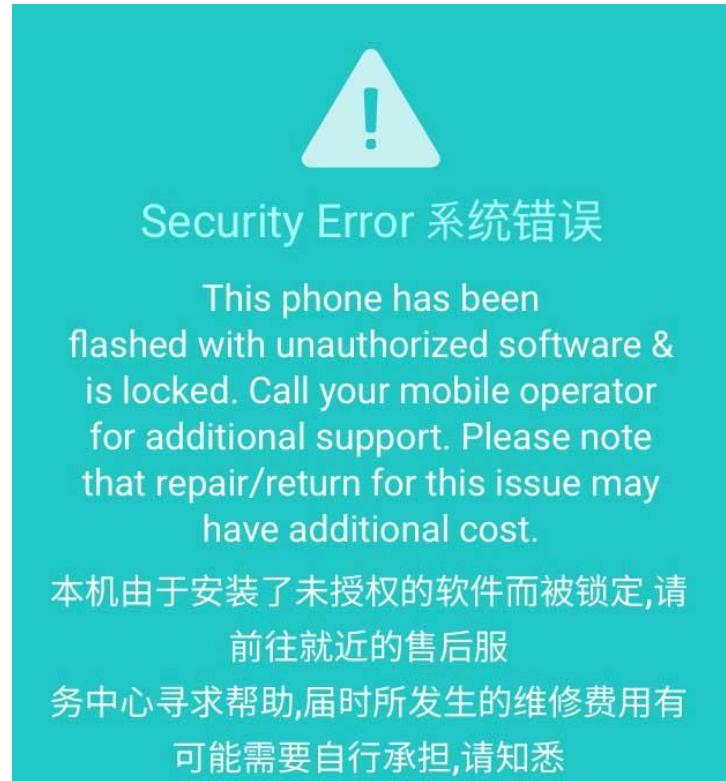


# Boot Process



# Little Kernel Patching

- Patching strategy: empirical approach
  - Reverse engineering and identify checks
  - Patch, test and repeat
- In the end we patch AVB to launch a modified boot image



# Little Kernel Patching

```
26 iVar1 = do_hash(param_1,param_2,DAT_4c6463e0 - param_2,&hash,0x20);
27 if (iVar1 == 0) {
28     iVar2 = memcmp(&STORED_HASH,&hash,0x20);
29     if (iVar2 == 0) {
30         print("[%s][oem] img auth pass\n",&s_SBC_030151a8);
31         goto LAB_02ff82e0;
32     }
33     iVar1 = 0x7021;
34 }
35 print("[%s][oem] img auth fail (0x%x)\n",&s_SBC_030151a8,iVar1);
```

# Little Kernel Patching

```
28 iVar1 = do_hash(param_1,param_2,_DAT_4c6463e0 - param_2,&hash,0x20);
29 if (iVar1 == 0) {
30     iVar2 = memcmp(&hash,&hash,0x20);
31     if (iVar2 == 0) {
32         print("[%s][oem] img auth pass\n",&DAT_030151a8);
33         goto LAB_02ff82e0;
34     }
35     iVar1 = 0x7021;
36 }
37 print("[%s][oem] img auth fail (0x%08x)\n",&DAT_030151a8,iVar1);
```

# Rooting Android

Main partitions used by Android: **boot** and **super**

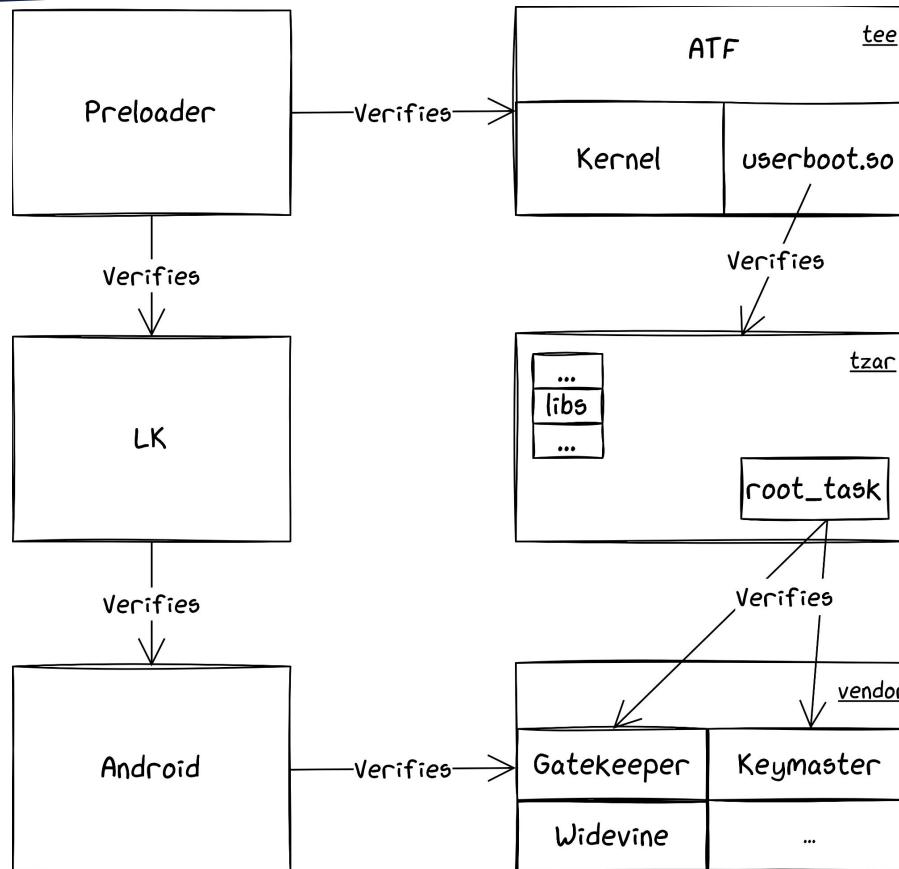
- Boot contains the kernel and a ramdisk (only used for first boot stage)
- Super is a Dynamic Partition that contains 4 logical partitions
  - system, vendor, product, odm

To root it

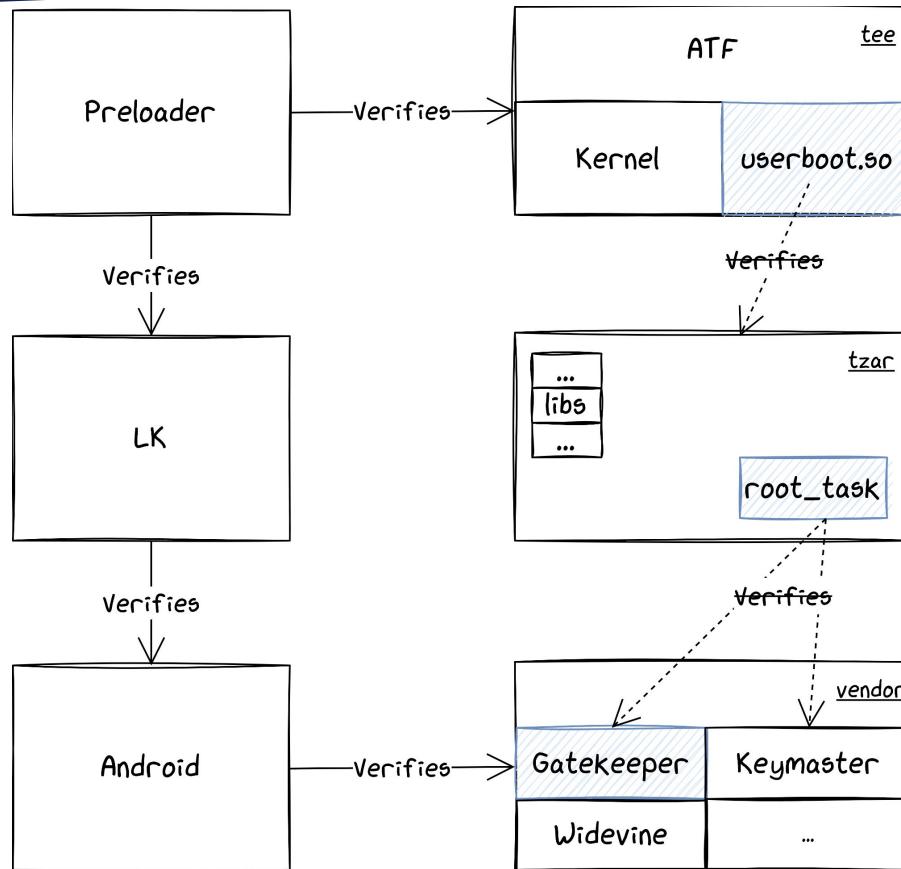
- Magisk<sup>5</sup> to patch the boot image
- We made few modifications to su
- Plus other little tricks to patch the super partition

- Trustzone OS designed by Samsung
- For Mediatek and Exynos SoCs
- ROM images:
  - tee1.img: ATF, TEEGRIS kernel, userboot.so
  - tzar.img: TEE root filesystem
  - super.img: Android system, Trusted Applications and Drivers
- Excellent references online<sup>6</sup>

# TEEGRIS Images Verification



# Patching TEEGRIS



# Reversing Gatekeeper

- TAs come in a slightly modified ELF format
  - 8-bytes header and footer with signature
  - Removing them allows to load a nice ELF in your favourite disassembler
- GlobalPlatform API
  - Standard API for TEEs (memory allocation, crypto operations, etc.)
  - Makes reversing easier
- Trusty reference implementation<sup>7</sup>
  - Suggests what to expect from a TA

# Gatekeeper Reference Implementation

- 2 Gatekeeper commands: Enroll and Verify
- Verify does two things:
  - $\text{HMAC}(\text{pwd\_handle}) == \text{expected}$ ?
  - If so, create new authentication token
- What if we can leak the key used by HMAC?
  1.  $\text{pwd} = \text{generate new password}$
  2.  $\text{Value} = \text{HMAC}(\text{pwd\_handle})$
  3.  $\text{Value} == \text{expected}$

# Reversing & patching Gatekeeper

- 2 Gatekeeper commands: Enroll and Verify
- Verify does two things:
  - HMAC(pwd\_handle) == expected?
  - If so, create new authentication token



# Reversing & patching Gatekeeper

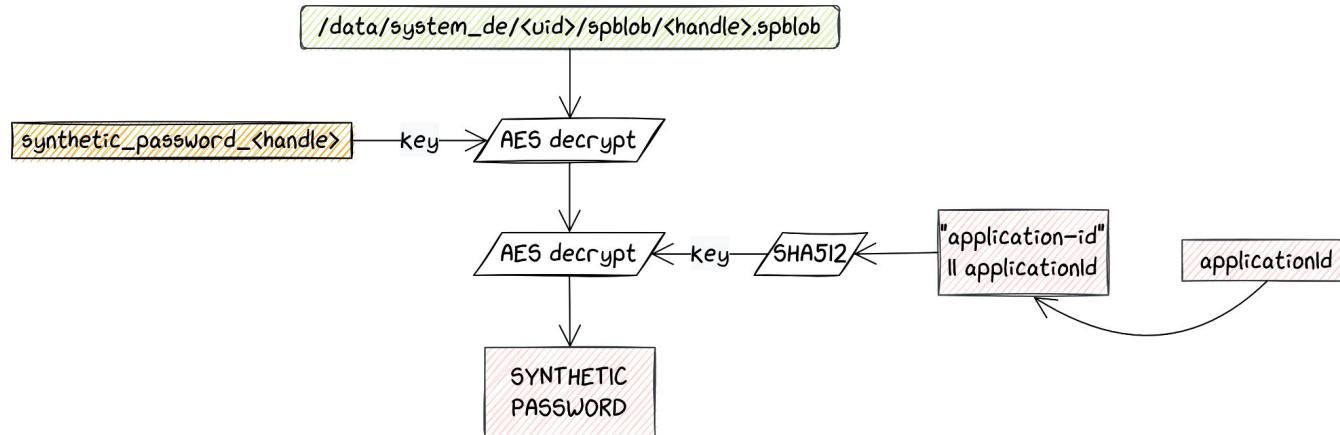
- This Gatekeeper implementation uses a KDF instead of a plain HMAC
  - KDF implemented in a library
  - which calls /dev/crypto
  - many steps to leak the key
- Simpler strategy: patch to accept any credentials
- Always return valid auth token to continue the process
  1. ~~KDF(pwd\_handle) == expected?~~
  2. ~~If so~~, create new auth\_token

# Reversing Gatekeeper

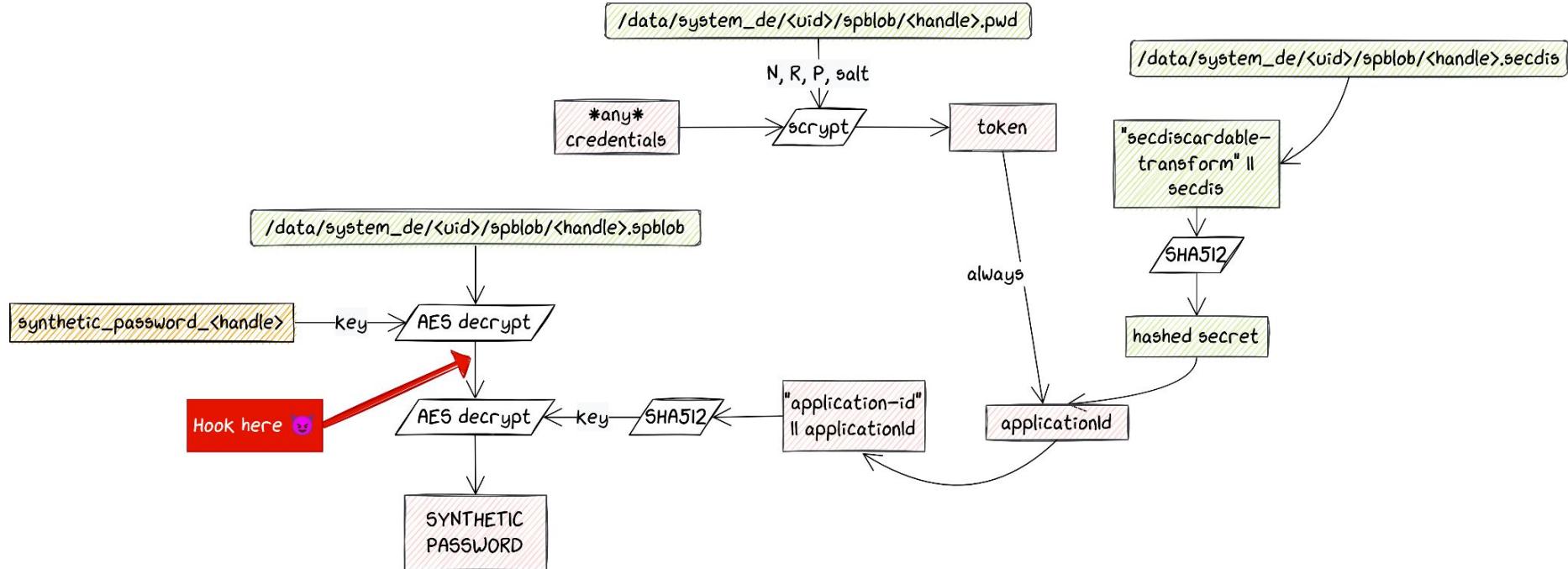
```
22 iVar1 = TEE_AllocateOperation(&local_30,0x50000004,5,0);
23 if (iVar1 == 0) {
24     iVar1 = TEE_DigestDoFinal(local_30,param_1,param_2,auStack_28,&local_38);
25     TEE_FreeOperation(local_30);
26     if (iVar1 == 0) {
27         uVar2 = TEE_AllocateTransientObject(0xa0000000,param_4 << 3,&local_30);
28         if (uVar2 == 0) {
29             uVar2 = TEES_DeriveKeyKDF(auStack_28,local_38,local_48,8,param_4,local_30);
30             if (uVar2 == 0) {
31                 uVar3 = 1;
32                 uVar2 = TEE_GetObjectBufferAttribute(local_30,0xc0000000,param_5,&iStack_34);
33                 if (uVar2 != 0) {
34                     uVar3 = 0;
35                     printf("gatekeeper [ERR] (%s:%u) failed to get object attribute: %x",
36                           "hal_pwd_hmac",
37                           0x12a,(ulong)uVar2);
38                 }
39             }
40         }
41     }
42 }
```

# Attack strategy

- Read the output of the first AES decrypt
- Bruteforce credentials to generate applicationId
- Thanks to GCM mode, AES decrypt complains if the key is wrong



# Hooking system\_server

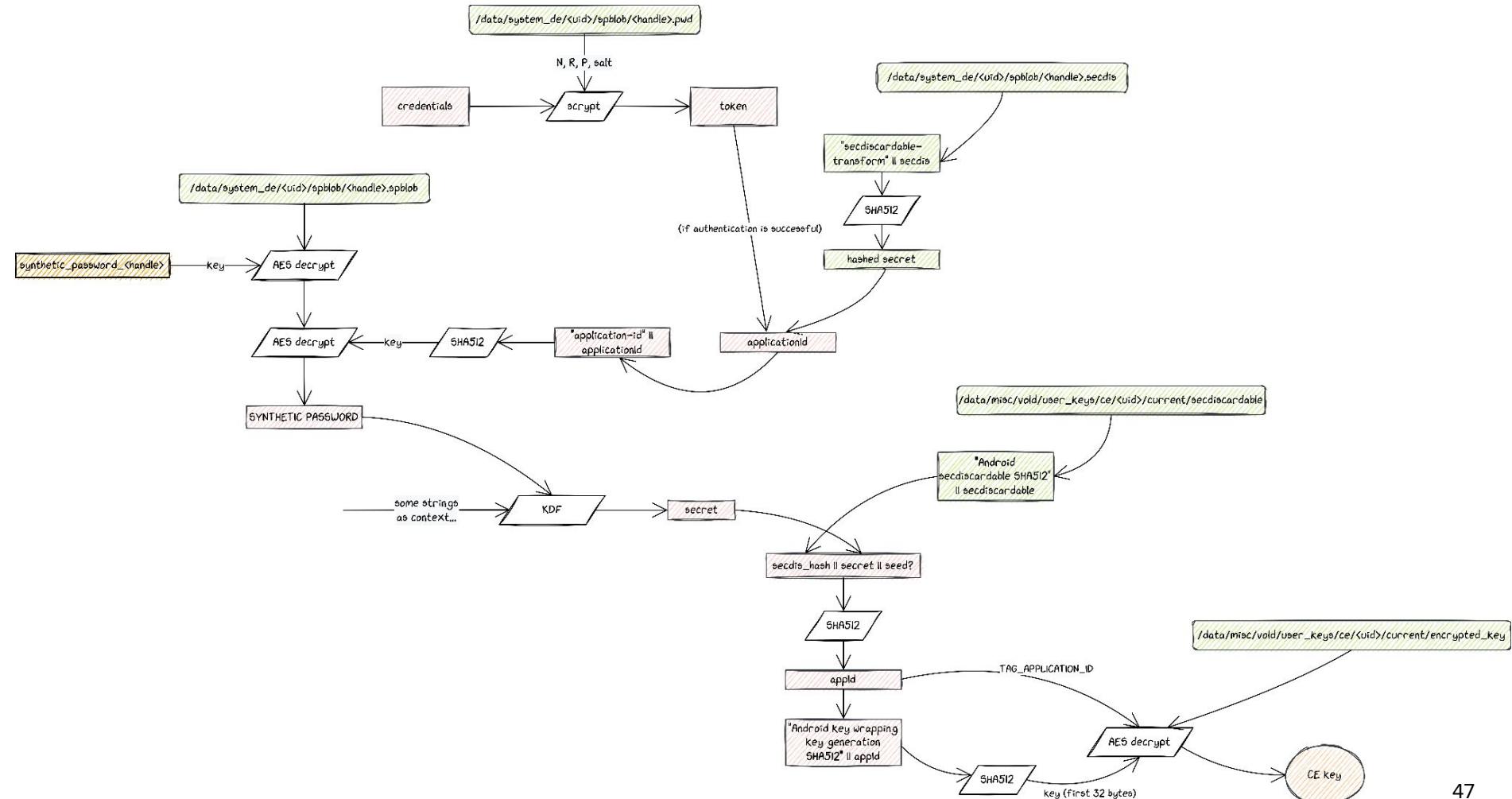


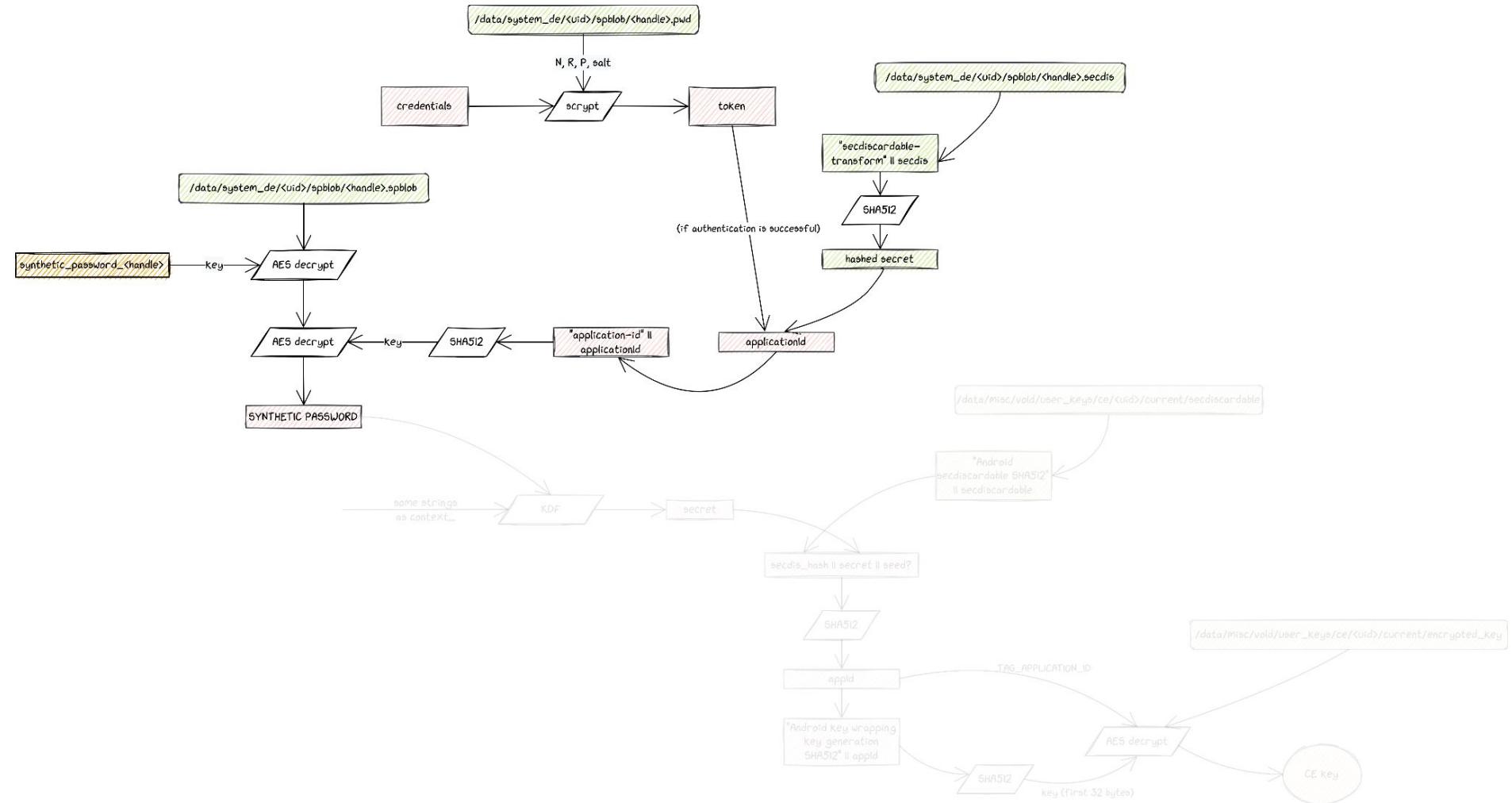
# Retrieving intermediate key with Frida

- Use Frida to hook system\_server
- Retrieve intermediate buffer decrypted by TEE
  - Possible thanks to the auth token

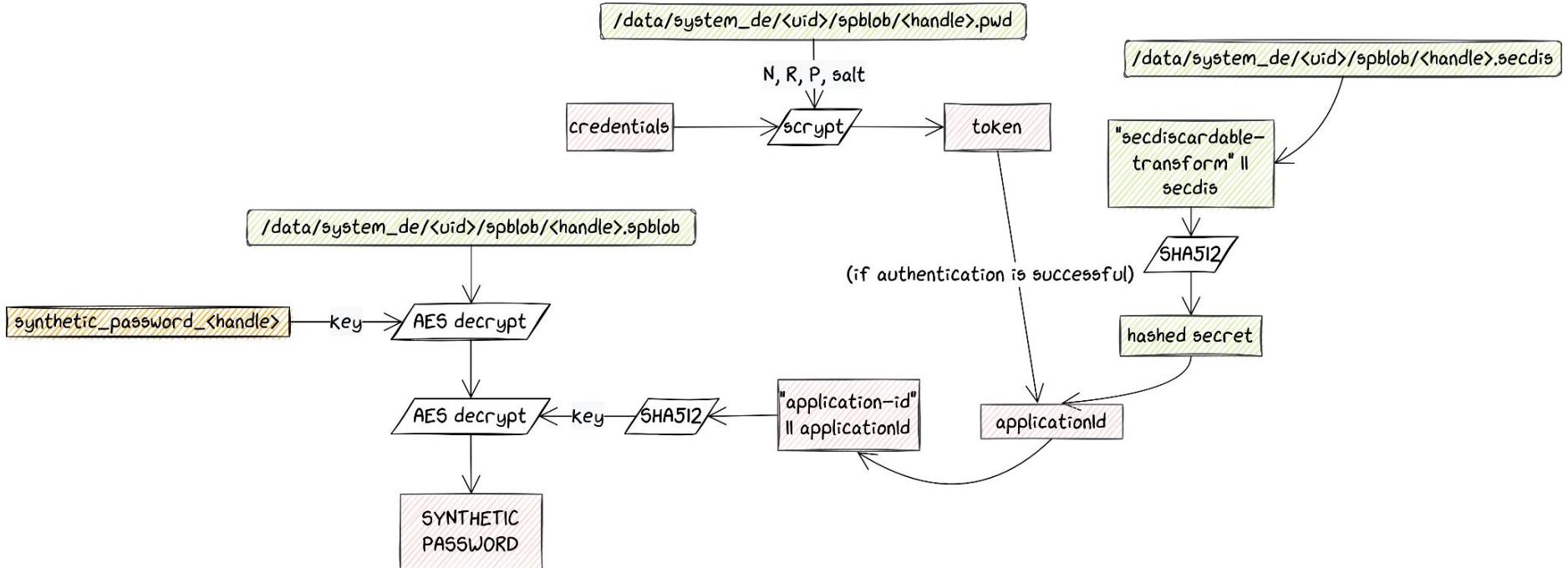
```
$ frida -U system_server -l system_server.js
    / _ _ |  Frida 16.0.19 - A world-class dynamic instrumentation toolkit
    | ( _ | |
    > _ _ |  Commands:
    /_/_ |  help      -> Displays the help system
    . . . . object?    -> Display information about 'object'
    . . . . exit/quit -> Exit
    . . . . More info at https://frida.re/docs/home/
    . . . . Connected to SM A226BR (id=R9WTA0BYDPL)

[SM A226BR::system_server ]-> SyntheticPasswordCrypto.decrypt called!
ciphertext = 641a3ed0a68abdae99976b5aff32f8d5aa4d18127272af6ff638c1e88d57cd157fd6f75b4688465f
470bd4cc81081215e9f2085e4b8ea22e0e8f0ed32a381f641d5cd071d2e177c4a8a1b6e6824f52f251366ff730f66
b7cfdf72f11f9761efc5e0cf68bd7bdec00456e07dfb9f1a7f720e97aa262c0507bc87ef46e603a265c821cb1a1dc5
c6f6be6fd43ac3431d0d013de8c9
[SM A226BR::system_server ]->
[SM A226BR::system_server ]-> |
```

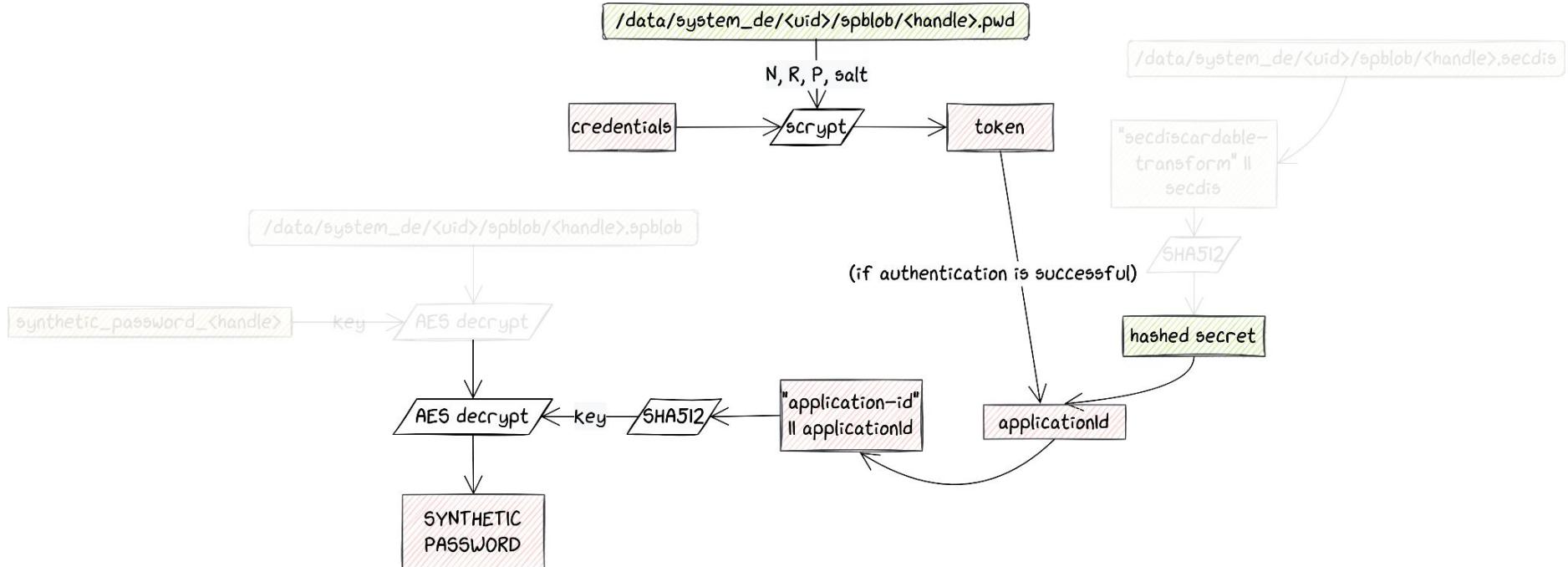




# Bruteforce of the password



# Bruteforce of the password



# Bruteforce of the password

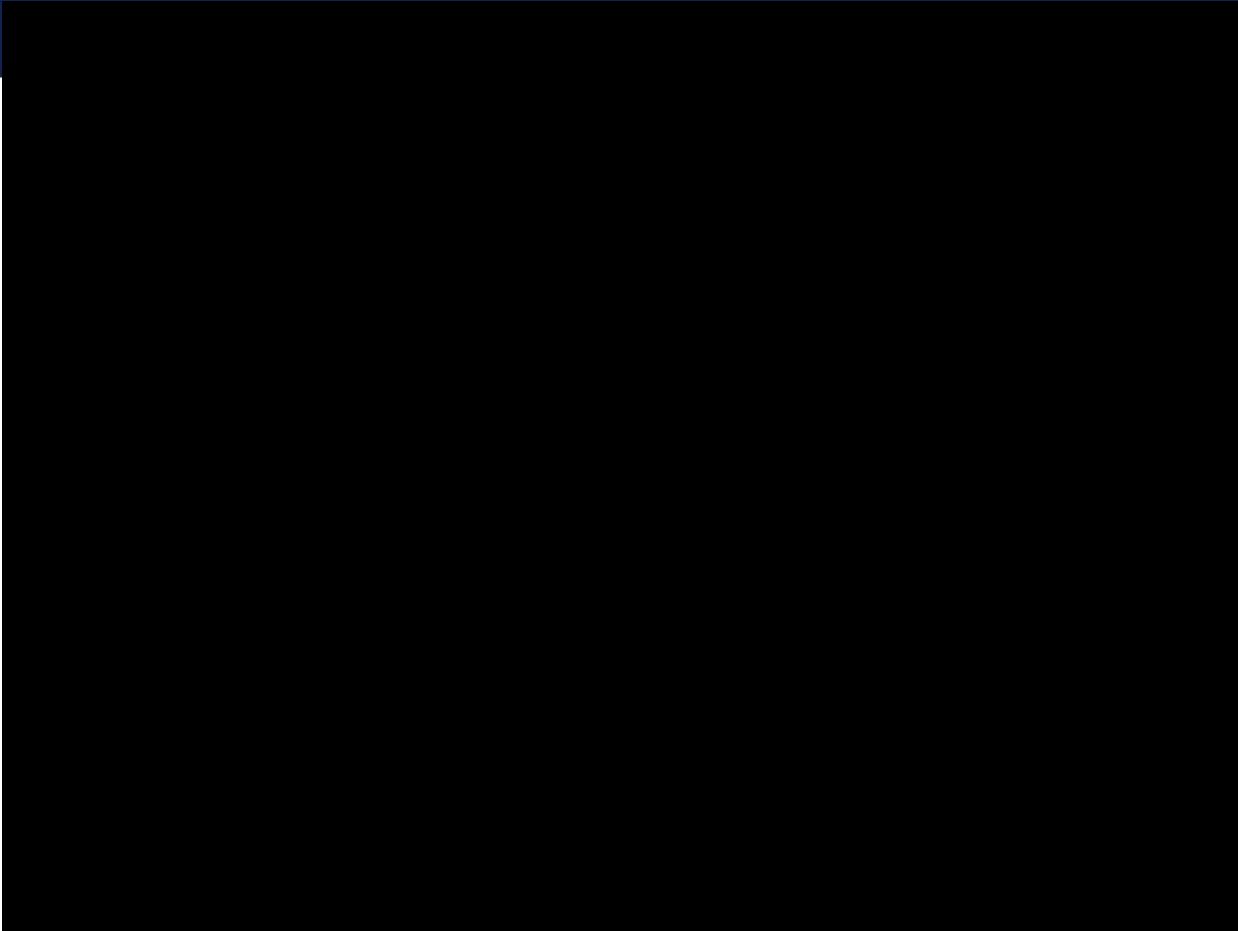
1. `pwd = generate new password`
2. `token = scrypt(pwd, R, N, P, Salt)`
3. `Application_id = token || Prehashed value`
4. `Key = SHA512("application_id" || application_id)`
5. `AES_Decrypt(value_from_keymaster, key)`

# Bruteforce of the password

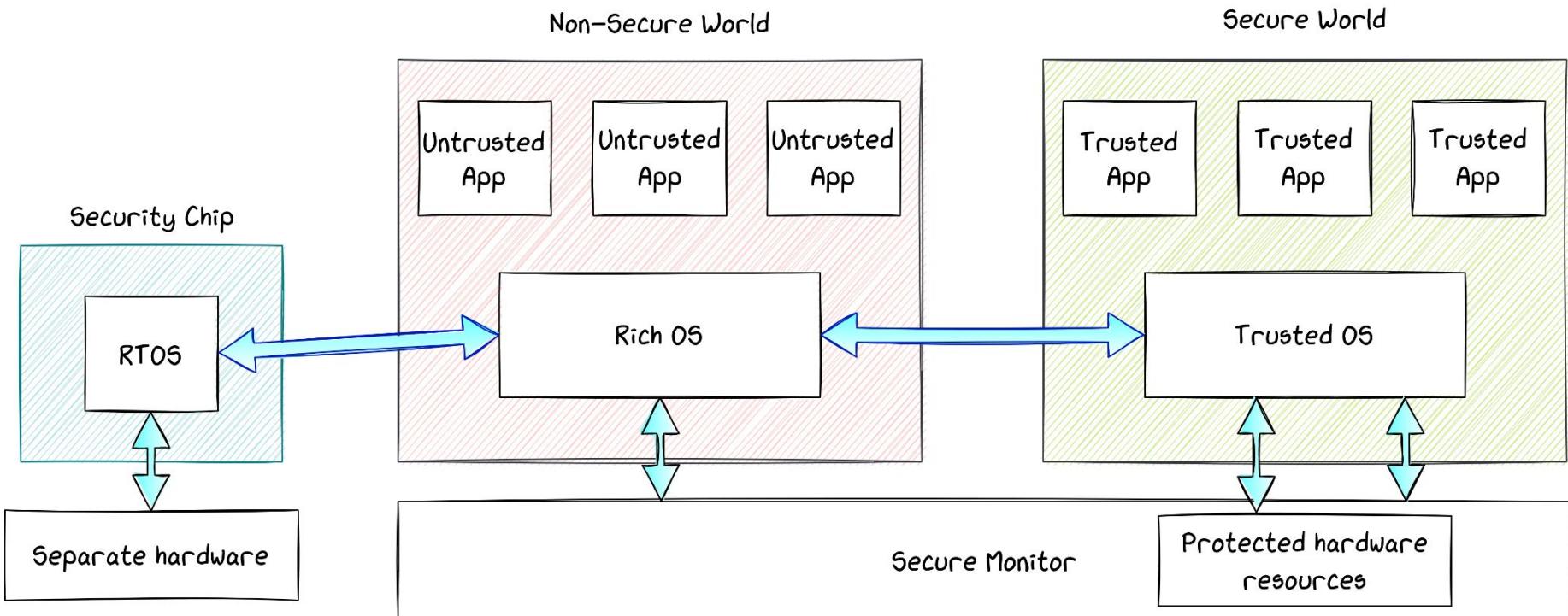
1. `pwd = generate new password`
2. `token = scrypt(pwd, R, N, P, Salt)`
3. `Application_id = token || Prehashed value`
4. `Key = SHA512("application_id" || application_id)`
5. `AES_Decrypt(value_from_keymaster, key)`

```
$ python3 bruteforce-tee.py
workers will cycle through the last 5 chars
Found it: 1234
the plaintext is '1234'
Done in 18.031058311462402s
Throughput: 1478.448992816657 tries/s
```

# Demo 1



# Architecture w/ Trusted Chip



# The Titan M Chip

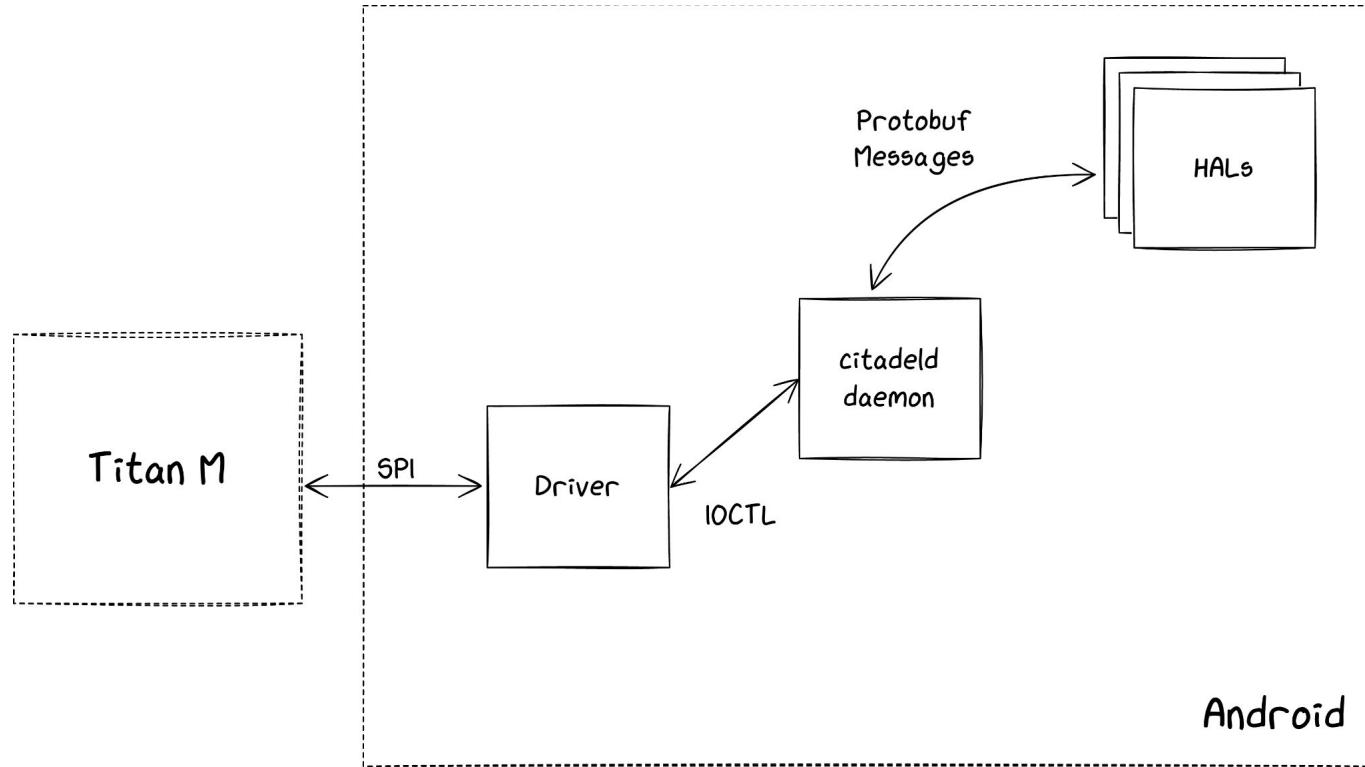
- Security chip made by Google for Pixel phones
- From Pixel 3 to Pixel 5a
  - In this PoC we use a Pixel 3a
  - Titan M2 introduced from Pixel 6
- Based on Arm Cortex-M3
- Most of the code is divided into tasks
  - Keymaster (Strongbox), **Weaver**, AVB, etc
- Separate memory and resources
  - Communicates with Android on SPI bus



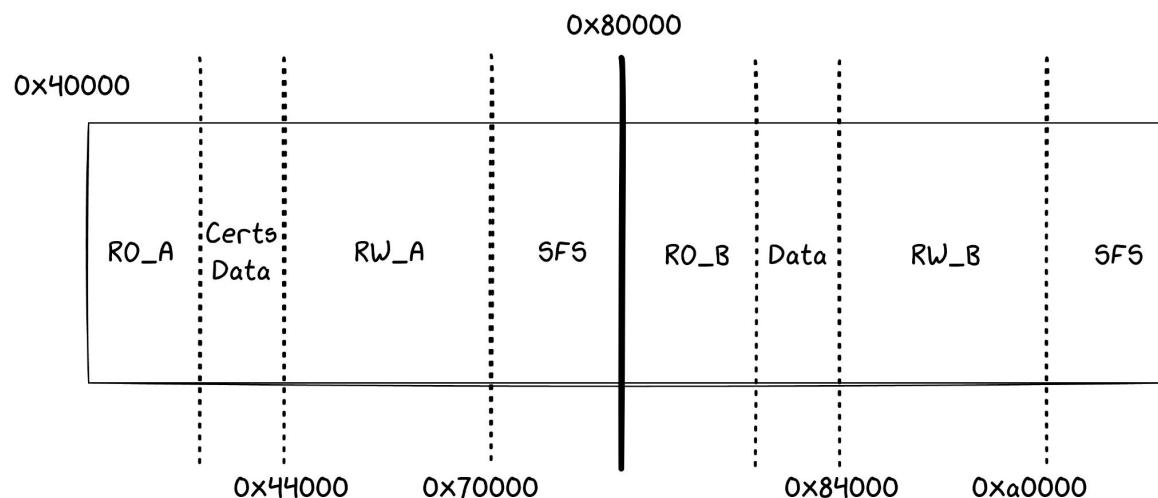
# Trusted chip vs TrustZone

- In TrustZone, secure and normal world run on the same CPU
  - Shared hardware (cache, RAM)
  - Side-channel attacks are possible (e.g. Rowhammer)
- Titan M relies on tamper-resistant hardware
- Separate firmware
  - Limited in size
  - Conceptually simple
  - Isolated from the rest of the system

# Communication with the chip



# Memory Layout



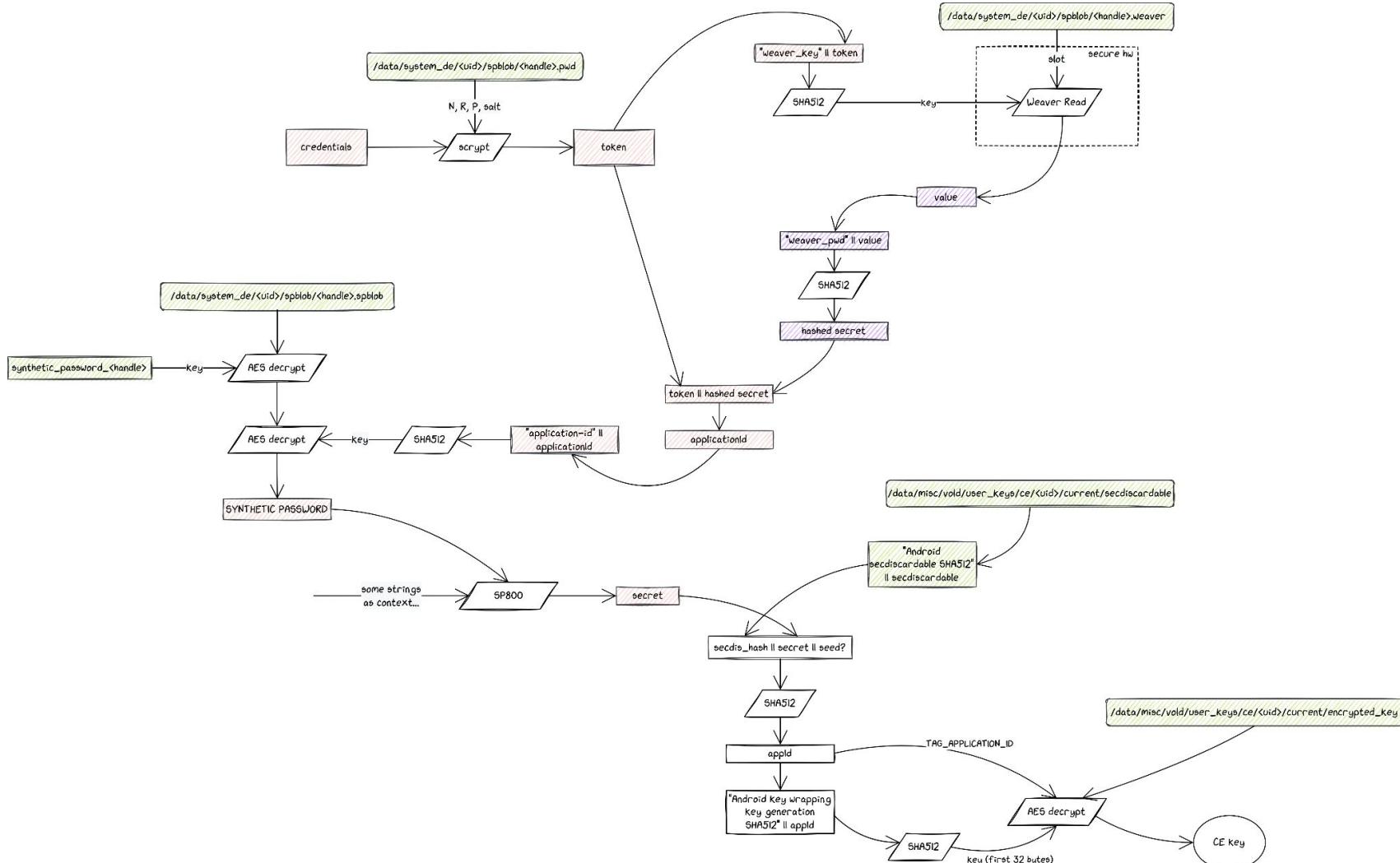
# Weaver

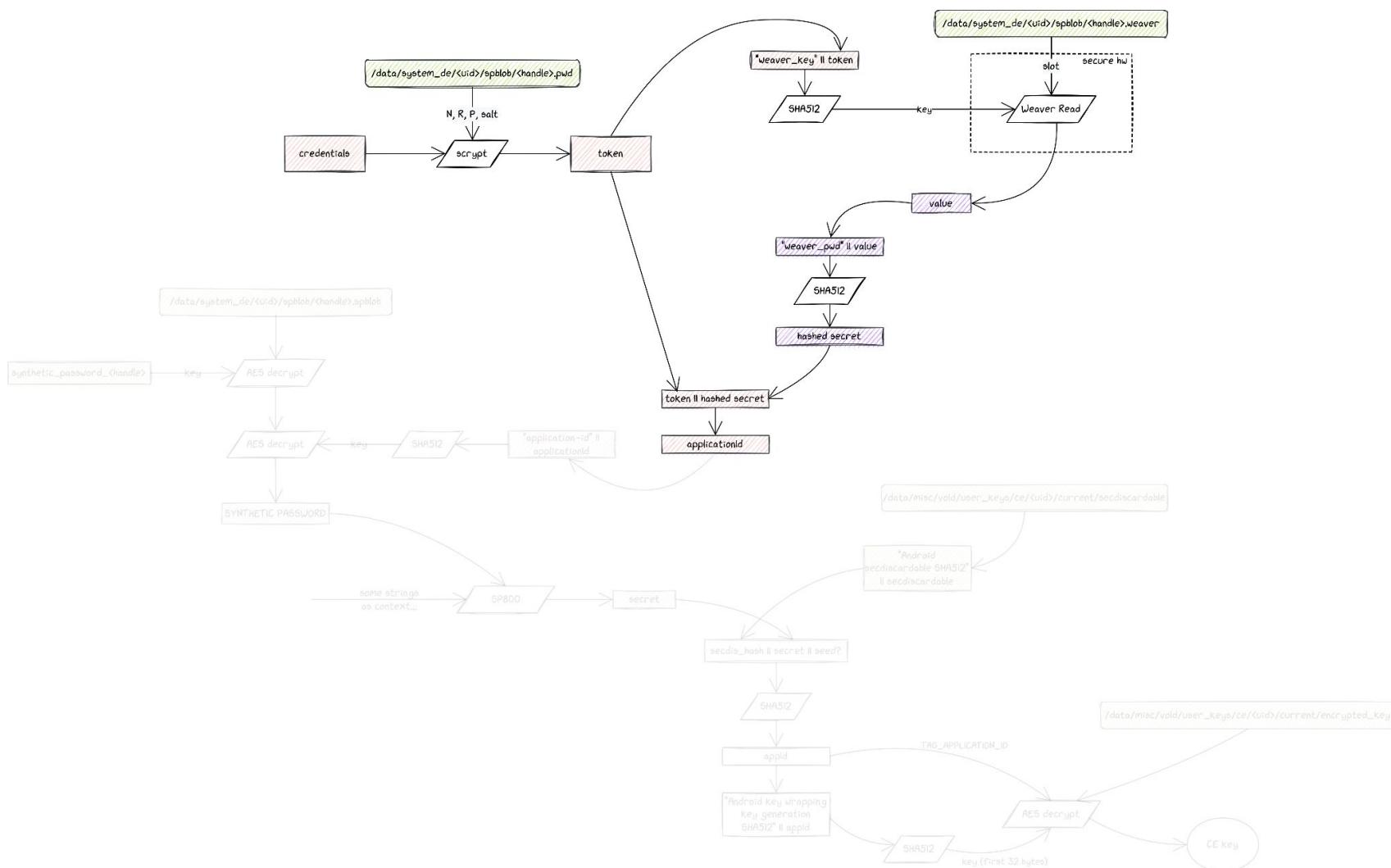
- Key/Value storage
  - Stored in slots
  - In two different places in the flash memory
- 4 commands: GetConfig, Read, Write, Erase
- Implements throttling as well

```
// Read

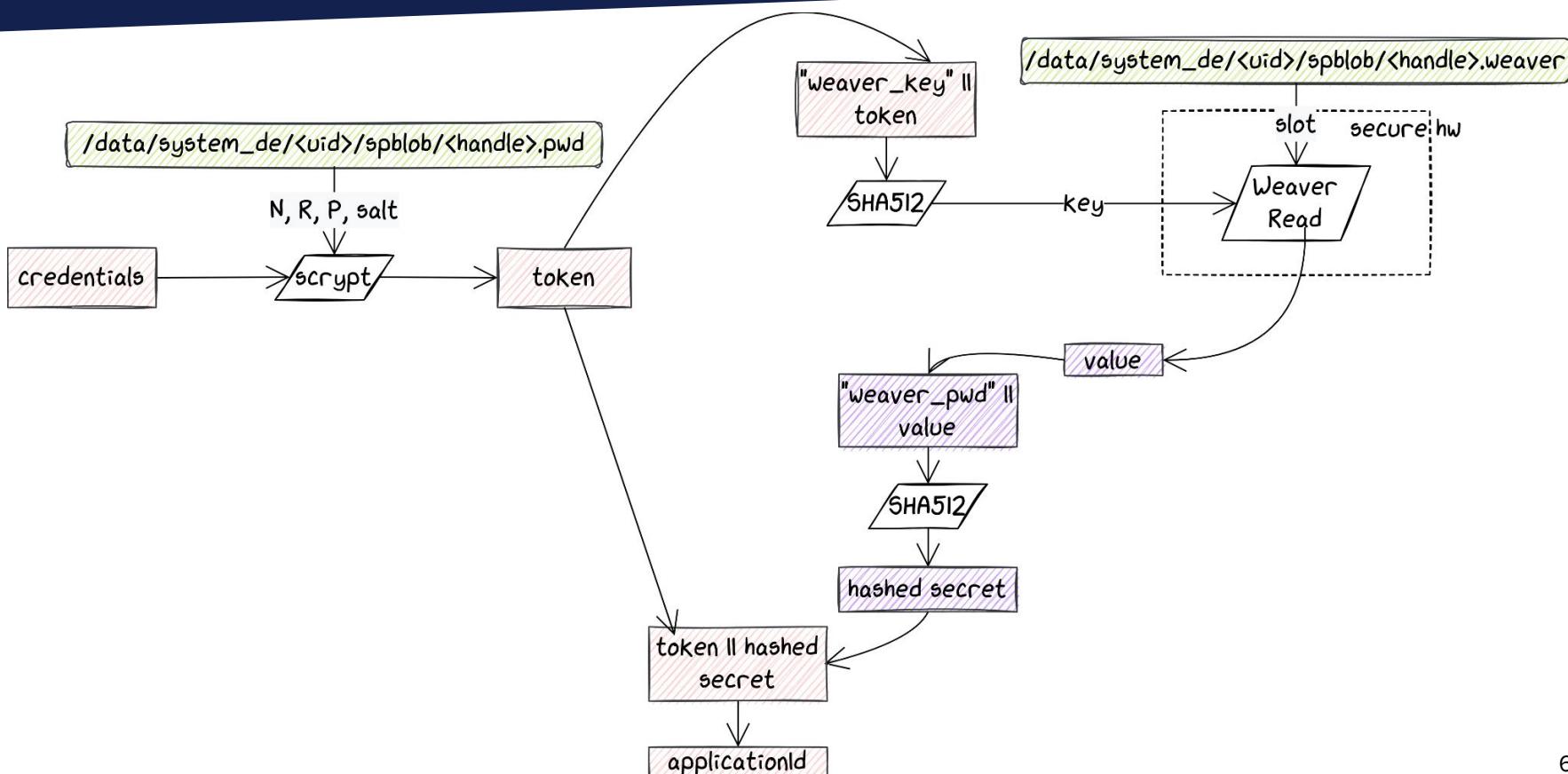
message ReadRequest {
    uint32 slot = 1;
    bytes key = 2;
}

message ReadResponse {
    Error error = 1;
    uint32 throttle_msec = 2;
    bytes value = 3;
}
```





# CE key derivation with Weaver



# PoC on Google Pixel

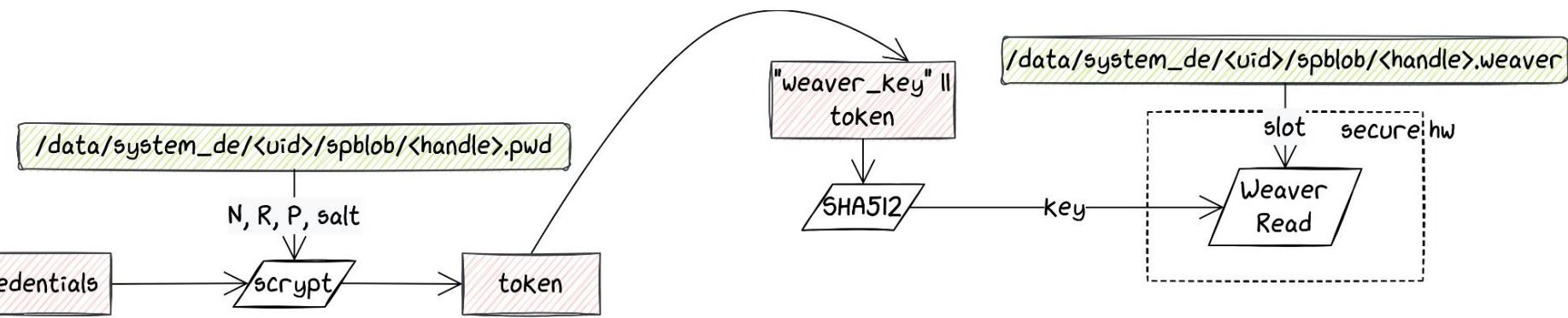
- We consider the device being already rooted
- Weaver relies on the security chip Titan M
- Here we exploit CVE-2022-20233 to execute code on the chip
- Out-of-bounds write of 1 byte to  $0x1$ 
  - Can be repeated multiple times
  - Huge constraints on the offset
  - We managed to overwrite a global field and cause another corruption
- Full exploit write-up in our blog<sup>8</sup>

[8]: <https://blog.quarkslab.com/attacking-titan-m-with-only-one-byte.html>

# Nosclient and the leak command

- We built a client to communicate with Titan M, nosclient
- “Leak” feature:
  - `./nosclient leak <address> <size>`
  - Read `<size>` bytes from `<address>`
  - Arbitrary read in Titan M’s memory
- Weaver slots and values are stored in flash
  - Reverse engineering to understand a memory range
  - Then search for 16 bytes digests
  - Weaver Write and Read help out

# Our Strategy



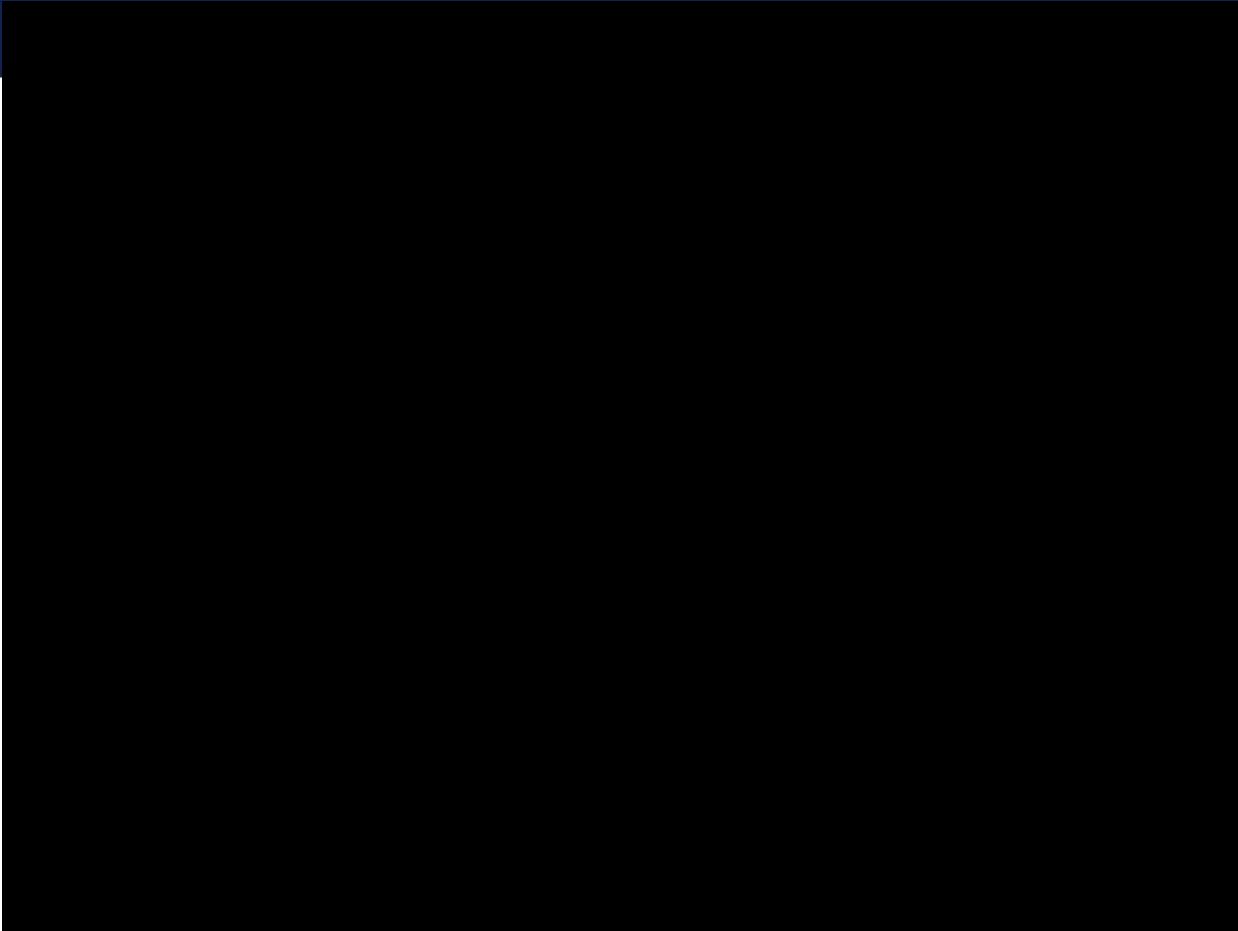
1. Leak the Weaver key
2. Use it to compare our generated credentials

# Bruteforce of the password

1. `pwd = generate new password`
2. `token = scrypt(pwd, R, N, P, Salt)`
3. `key = SHA512("weaver_key" || token)`
4. Compare with leaked Weaver key

```
$ python3 bruteforce.py
workers will cycle through the last 5 chars
Found it: 1106
the plaintext is '1106'
Done in 15.063793659210205 s
Throughput: 1491.722504195411 tries/s
```

## Demo 2



# Conclusion

- FBE is very well designed
- Ingredients from “everywhere” are used to derive the key
  - Files owned by privileged users
  - TEE-protected keys
  - Weaver values (when available)
- Multiple bugs needed to break it
  - Or a very powerful one
- You still need to bruteforce credentials in the end
- “my very secret password example for REcon 2023” will be hard to guess :)

# Thank you!

contact@quarkslab.com

Quarkslab



@DamianoMelotti  
@max\_r\_b

# Little Kernel

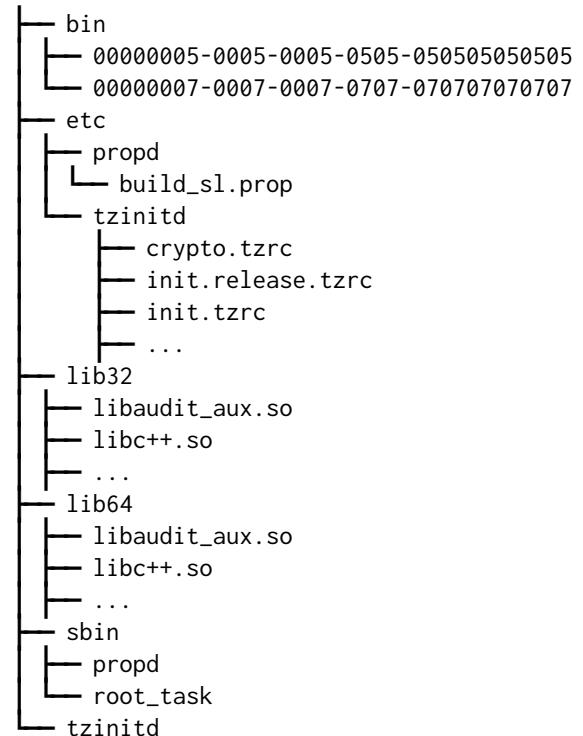
LK: Android bootloader based on Little Kernel

- Allows to boot Android or other modes (Recovery)
- Loads TZAR image in TEEGRIS
- Implements **Android Verified Boot v2**
  - Verification of Android images
  - Involving boot and vbmeta images
  - Anti-rollback

# TZAR image

TrustZone ARchive: contains a root filesystem

- Shared libraries
- Binaries
- tzinitd (init binary)
- root\_task



# Patching TEEGRIS

Our final goal is to run a modified Gatekeeper TA

- Patch userboot.so from the tee1 partition
  - Disable verification of TZAR image
- Patch root\_task from the TZAR image
  - Disable verification of TA
- Patch the Gatekeeper TA
  - Accept any credentials and return a valid auth token