TEL AVIV UNIVERSITY

Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Cryptographic Design

Alon Shakevsky, Eyal Ronen, Avishai Wool

@shakevsky @eyalr0 yash@eng.tau.ac.il

Extended paper: https://eprint.iacr.org/2022/208.pdf

Tool + PoC: https://github.com/shakevsky/keybuster

WHO WOULD WIN?

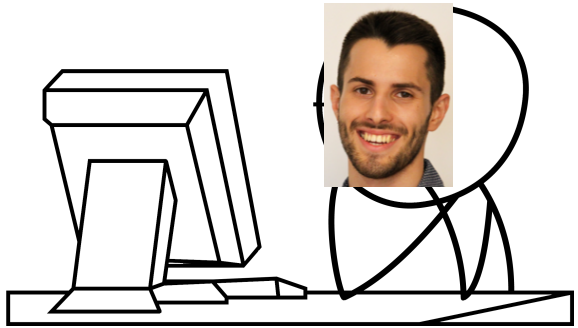
The leading Android Vendor

SAMSUNG

Common Criteria

FIPS LEVEL 1 VALIDATED FIPS

3 academic researchers



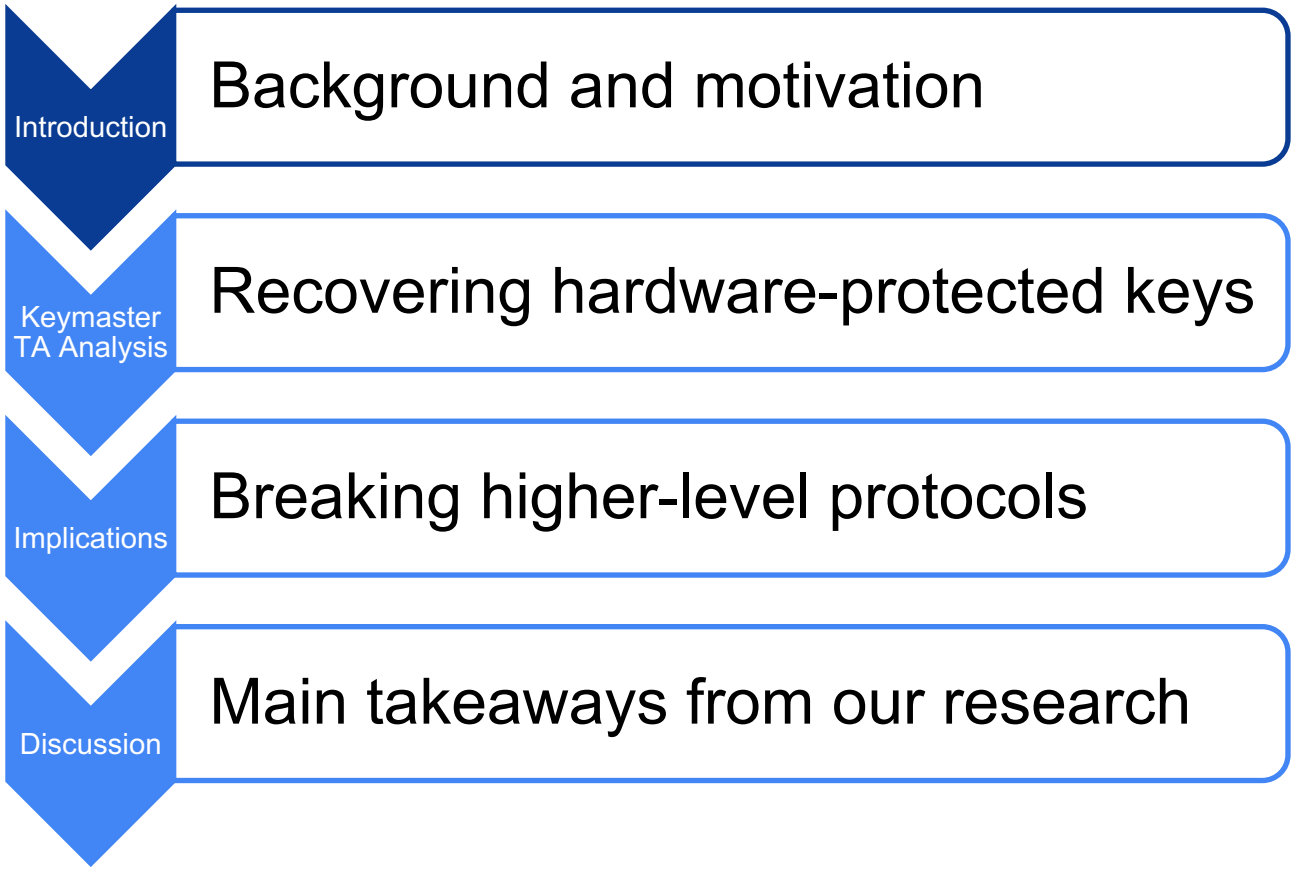
What did we find?

2 High severity CVEs that affect over 100 million devices

Recover keys that were encrypted by trusted hardware

Designed using resources from Flaticon.com

Agenda


Introduction
Background and motivation
Keymaster TA Analysis
Recovering hardware-protected keys
Implications
Breaking higher-level protocols
Discussion
Main takeaways from our research

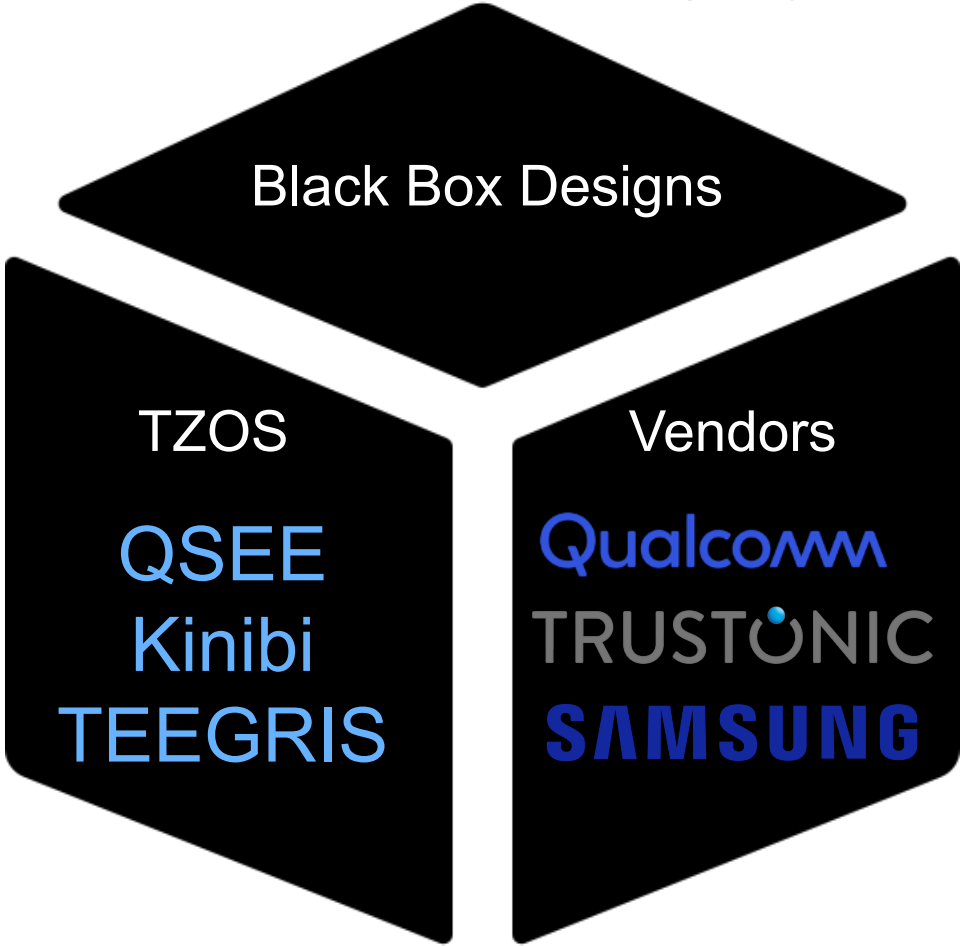
The need for Trusted Execution Environments (TEEs)

Designed using resources from Flaticon.com

The need for Trusted Execution Environments (TEEs)

Designed using resources from Flaticon.com

Proprietary TrustZone Operating Systems (TZOS)



Designed using resources from Flaticon.com

Research questions

1. Do hardware-protected cryptographic keys remain secure even when the Normal World (Android) is compromised?

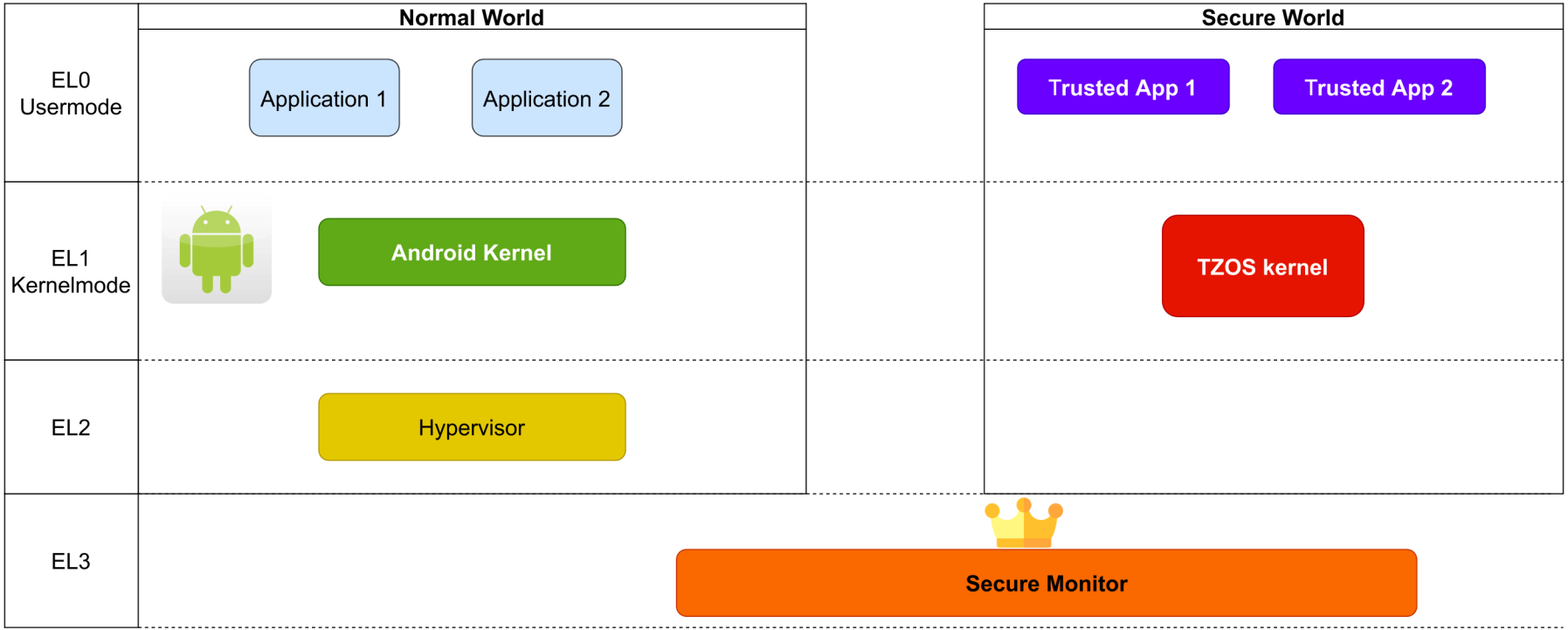
Designed using resources from Flaticon.com

Research questions

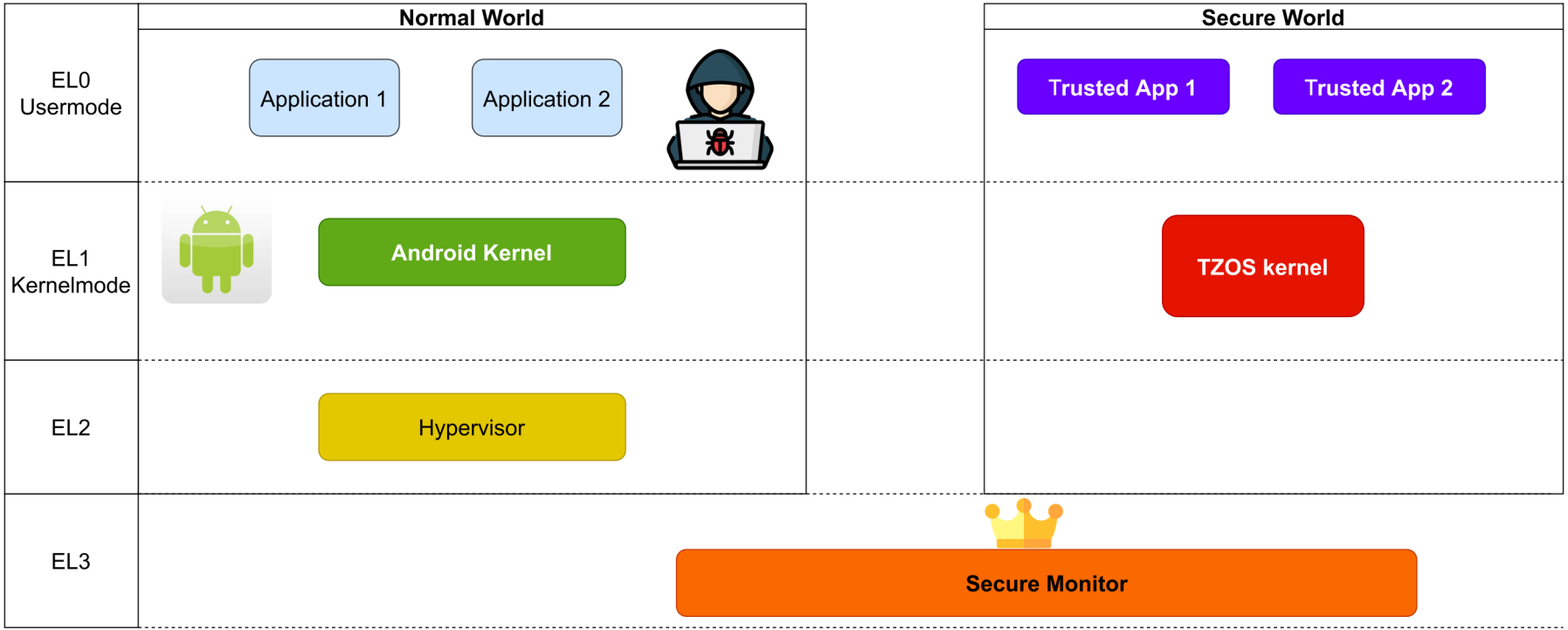
1. Do hardware-protected cryptographic keys remain secure even when the Normal World (Android) is compromised?
2. Do compromised hardware-protected keys break the security of various protocols that rely on them?

Designed using resources from Flaticon.com

ARM TrustZone - Attack Model

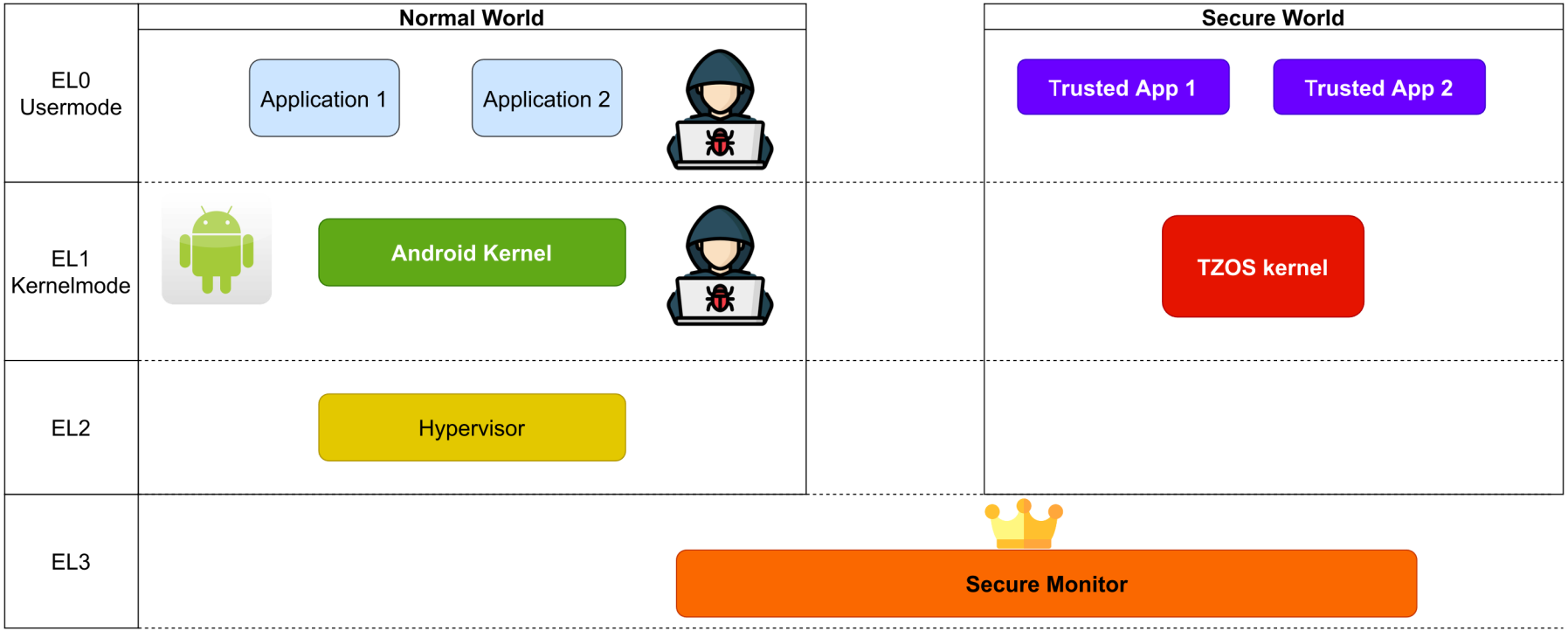


ARM TrustZone - Attack Model



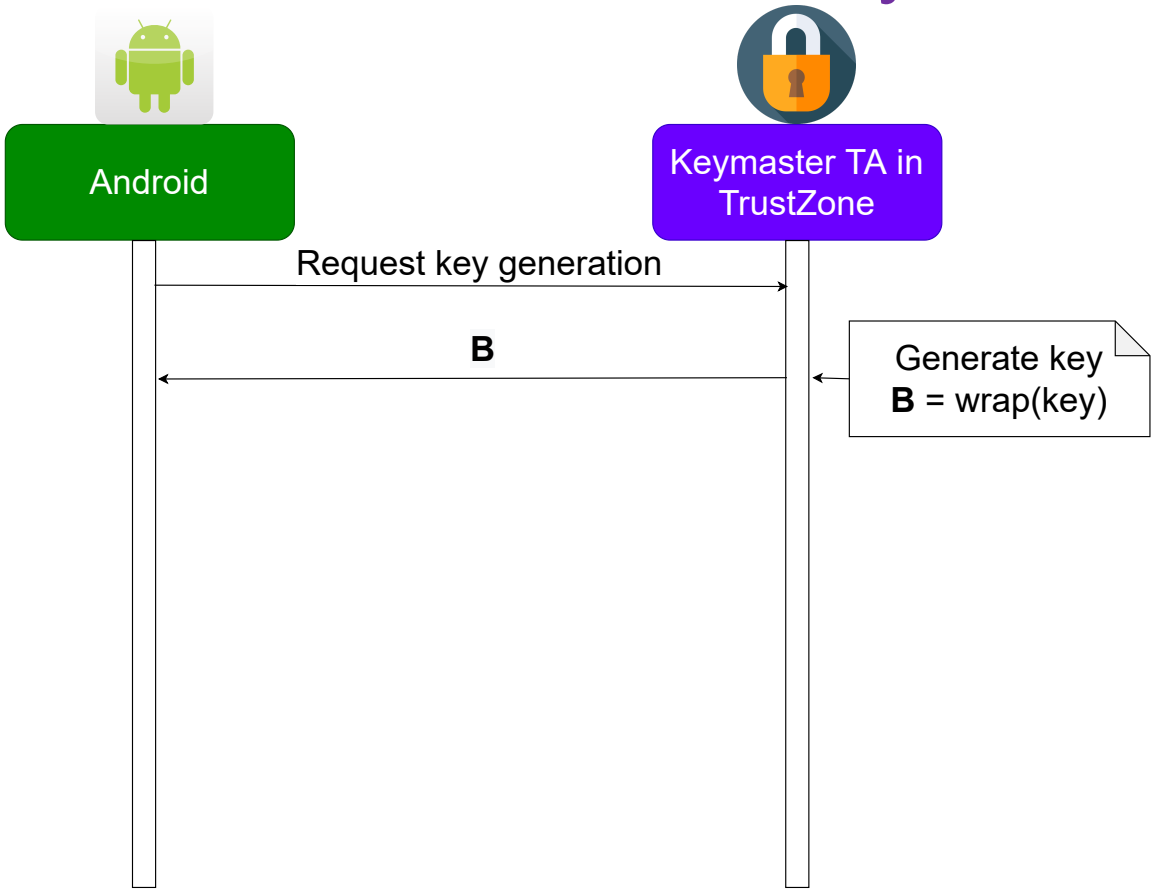
Designed using resources from Flaticon.com

ARM TrustZone - Attack Model



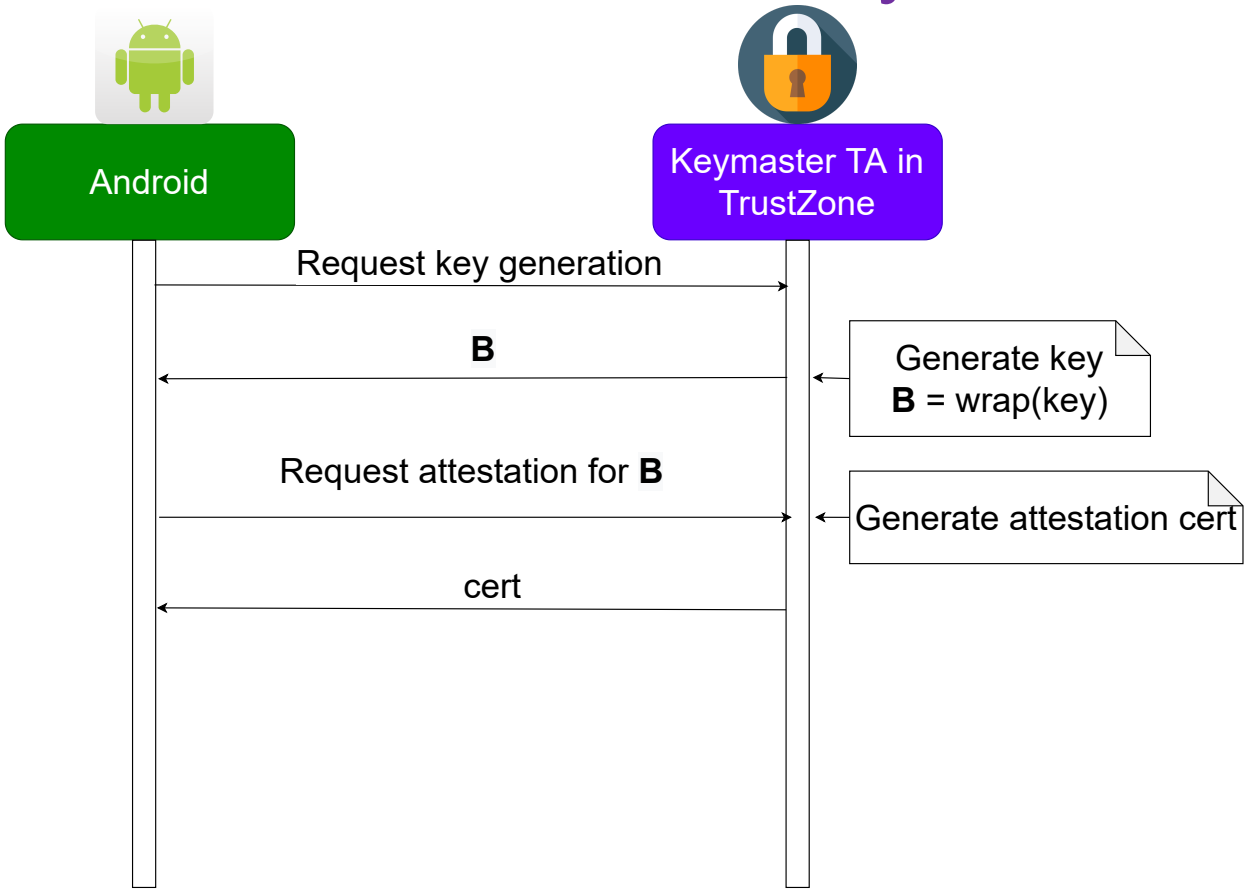
Designed using resources from Flaticon.com

Android Hardware-Backed Keystore Flow



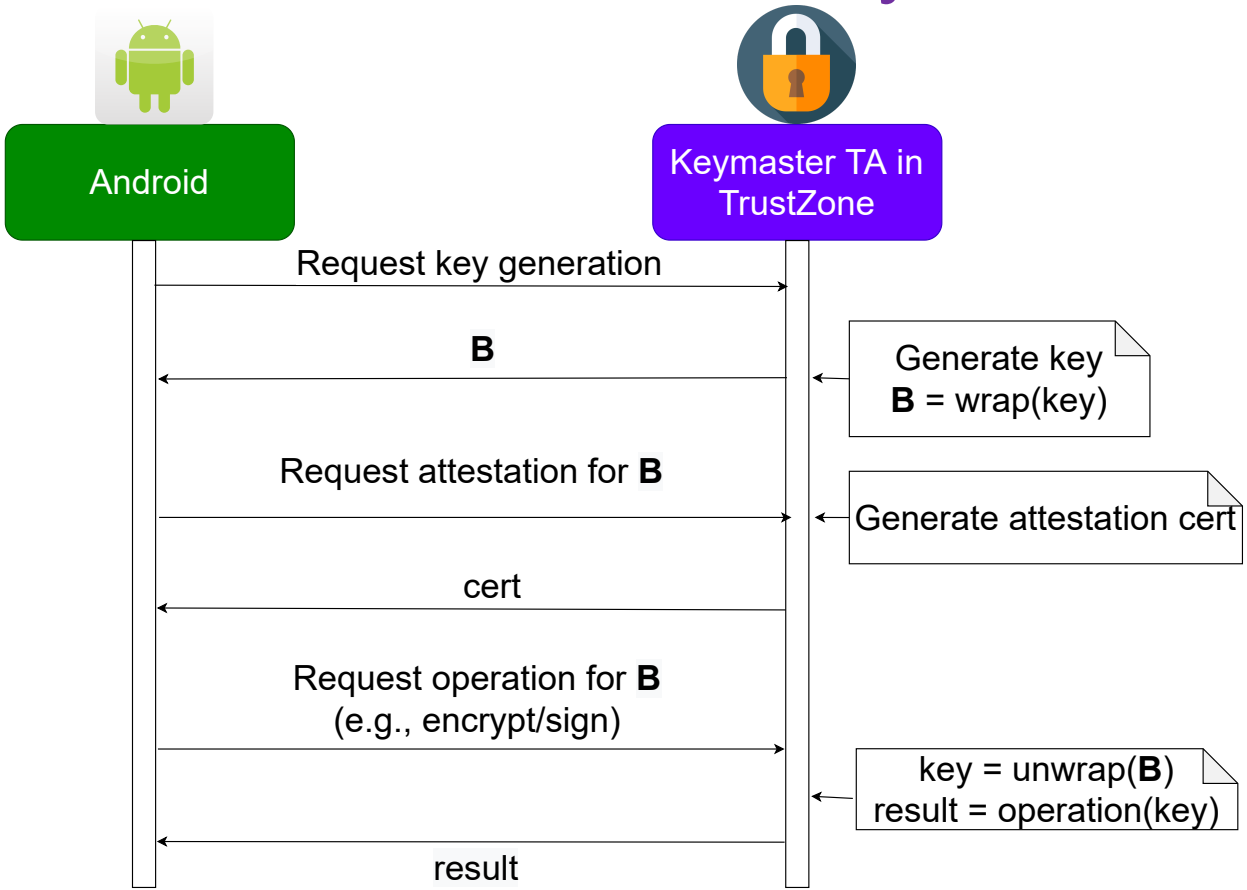
Designed using resources from Flaticon.com

Android Hardware-Backed Keystore Flow

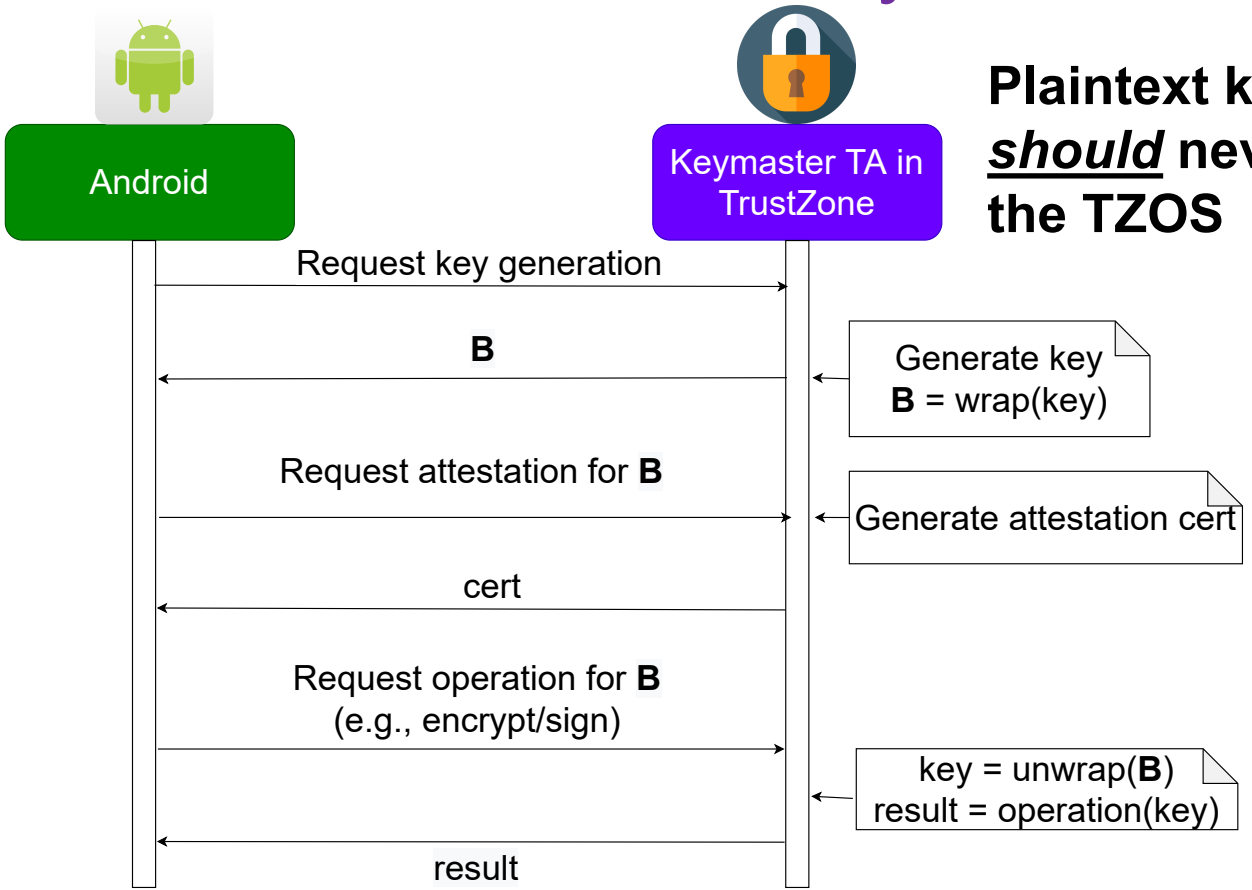


Designed using resources from Flaticon.com

Android Hardware-Backed Keystore Flow



Android Hardware-Backed Keystore Flow



Plaintext key material *should* never leave the TZOS

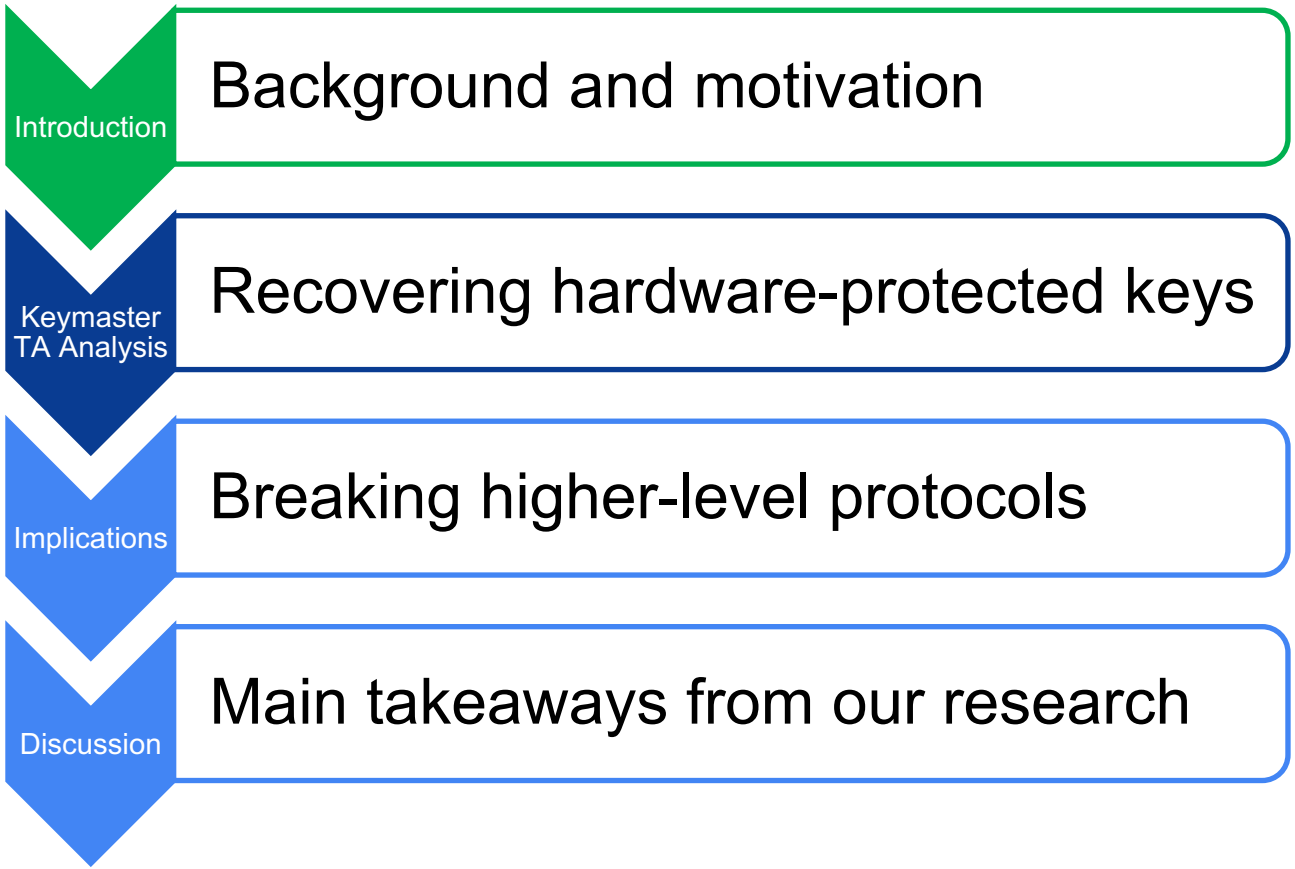
What’s the context?

We need to protect cryptographic keys of applications

Only the Keymaster should access key material

But is it guaranteed?

Agenda


Introduction
Background and motivation
Keymaster TA Analysis
Recovering hardware-protected keys
Implications
Breaking higher-level protocols
Discussion
Main takeaways from our research

Disclaimer

Where do you start?

Where do you start?

Download the firmware of the specific model

Where do you start?

Download the firmware of the specific model

Read public documentation and security certifications

Where do you start?

Download the firmware of the specific model

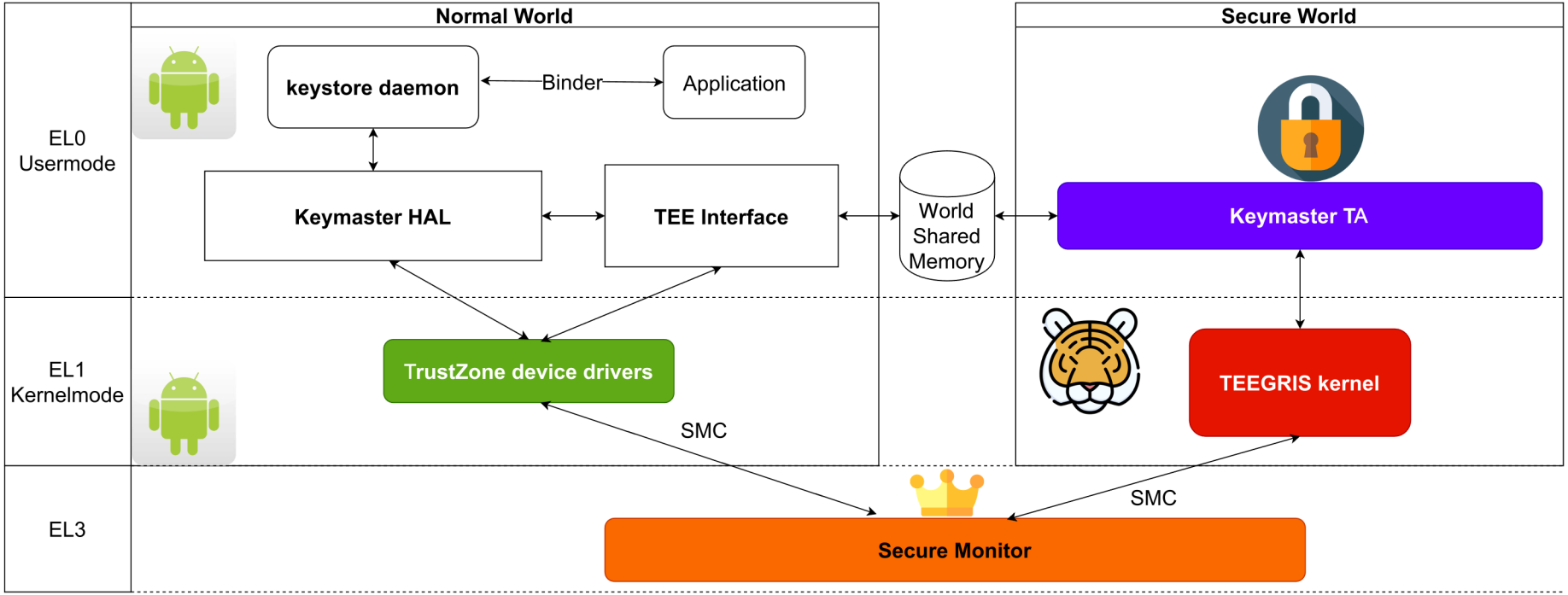
Read public documentation and security certifications

Reverse-engineer using Ghidra

Repeat for 26 firmwares

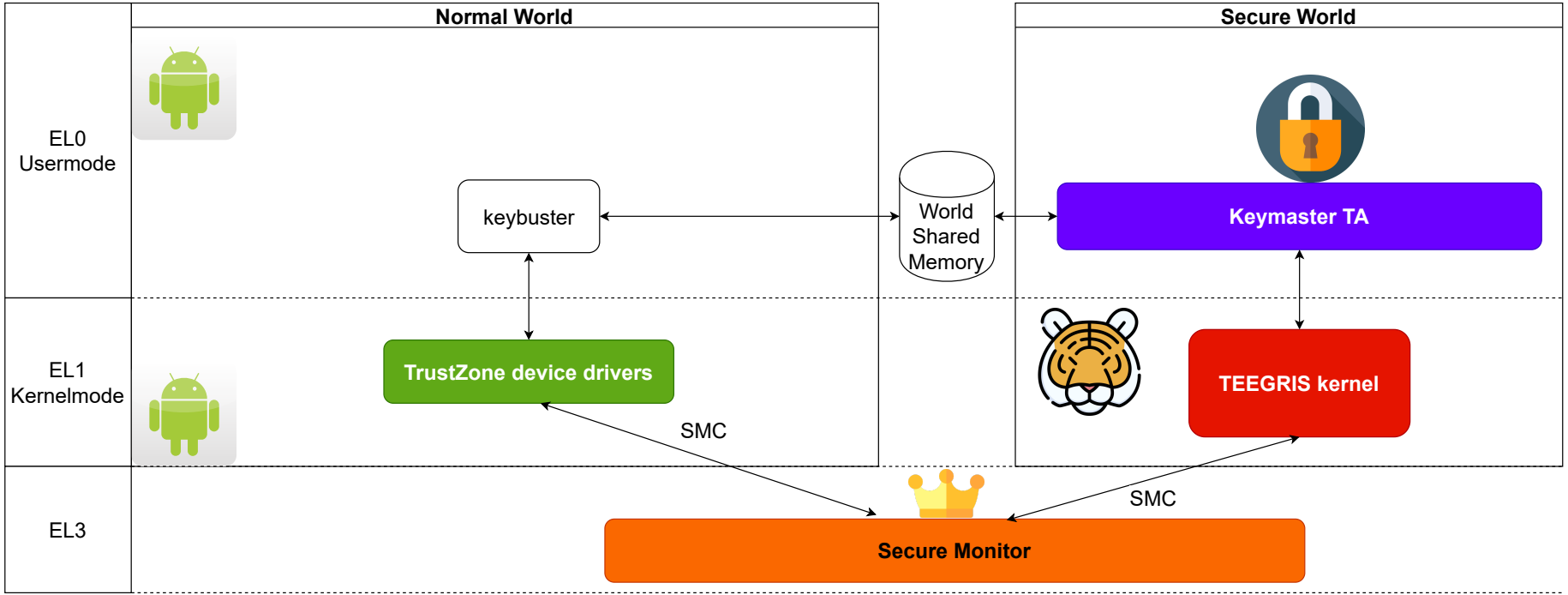
Image: Ryan Kurtz, Apache License 2.0 via Wikimedia Commons

How to interact with the Keymaster?



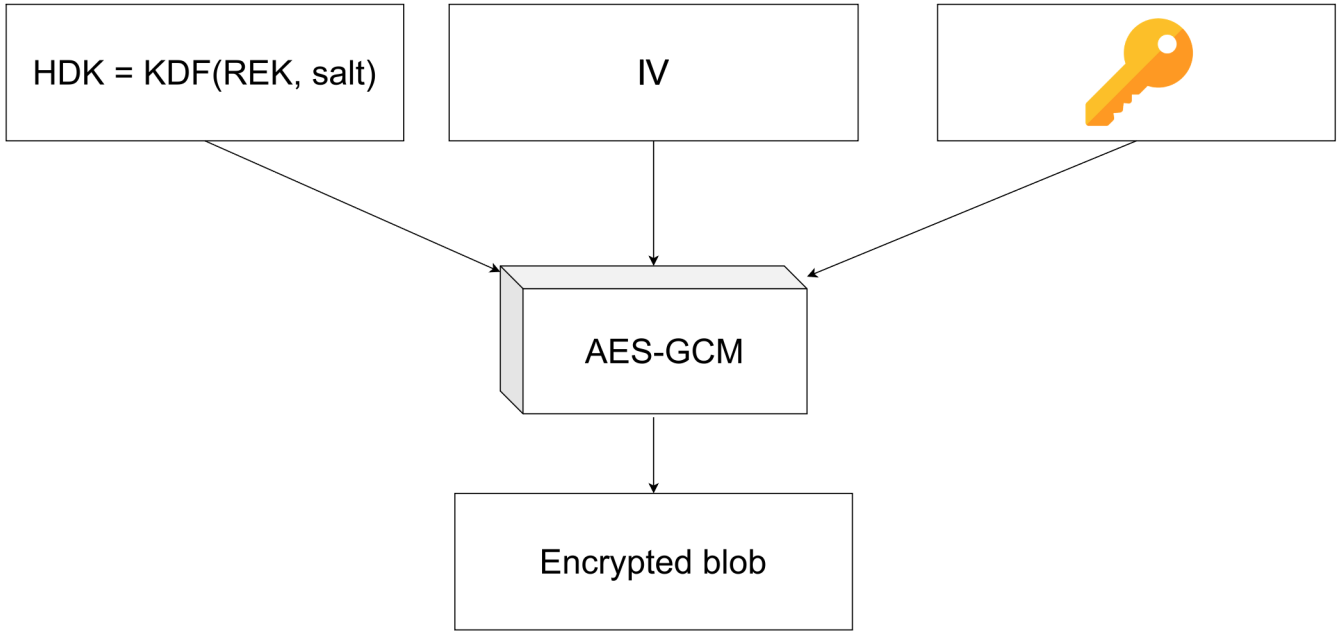
Designed using resources from Flaticon.com

Keybuster: tool to interact with the Keymaster



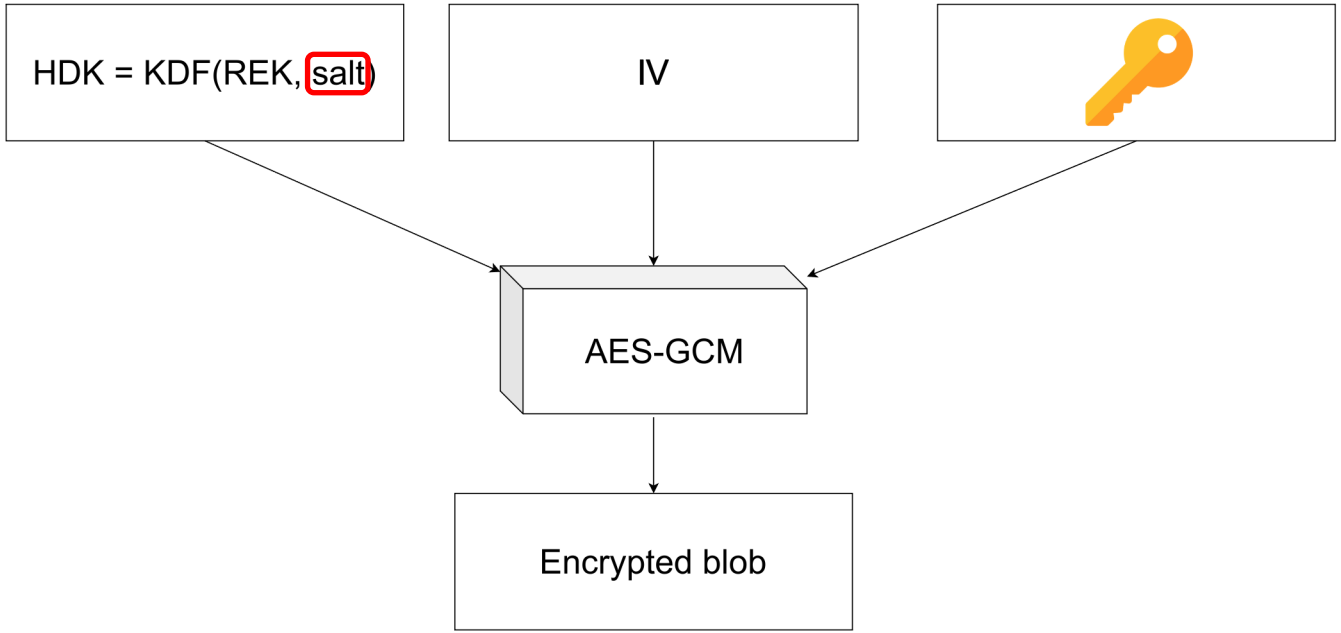
Key Blob Encryption

The Keymaster TA encrypts key material inside blobs.



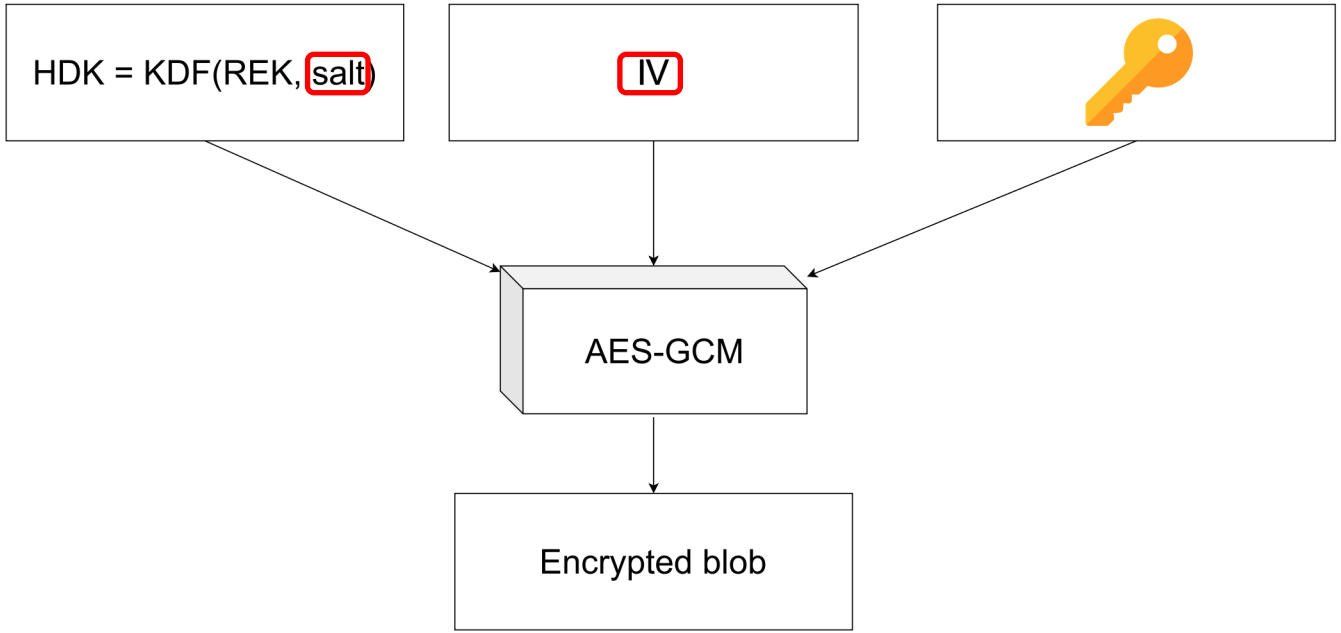
Key Blob Encryption

The Keymaster TA encrypts key material inside blobs.



Key Blob Encryption

The Keymaster TA encrypts key material inside blobs.



KDF versions of key blobs

salt = SHA-256(salt_seq)

Where salt_seq is one of the following sequences:

v15 blob
"MDFPP HW Keymaster HEK v15\x00"
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"

v20-s9 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags

v20-s10 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags
hek_randomness

KDF versions of key blobs

salt = SHA-256(salt_seq)

Where salt_seq is one of the following sequences:

v15 blob
"MDFPP HW Keymaster HEK v15\x00"
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"

v20-s9 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags

v20-s10 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags
hek_randomness

MDFPP can explain the variations

SAMSUNG

Common Criteria and FIPS-validated devices for the security conscious.

Common Criteria-Certified Devices, MDFPP v3

Supported on Android 10

- Samsung Galaxy Note20 5G | Note20 Ultra 5G
- Samsung Galaxy Tab S7 | Tab S7+
- Samsung Galaxy S20 FE 5G
- Samsung Galaxy S20 5G | S20+ 5G | S20 Ultra 5G
- Samsung Galaxy S20 Tactical Edition
- Samsung Galaxy Z Flip | Z Flip 5G
- Samsung Galaxy XCover Pro
- Samsung Galaxy XCover FieldPro
- Samsung Galaxy A71 5G
- Samsung Galaxy A51 I A51 5G
- Samsung Galaxy S10e | S10 | S10+ | S10 5G
- Samsung Galaxy Note10 | Note10+ | Note10+ 5G
- Samsung Galaxy Fold | Fold 5G
- Samsung Galaxy Z Fold2
- Samsung Galaxy S9 | S9+
- Samsung Galaxy Note9
- Samsung Galaxy Tab S6 | Tab S6 5G
- Samsung Galaxy Tab Active3
- Samsung Galaxy Tab S4

IV Reuse Attack (v15/v20-s9)

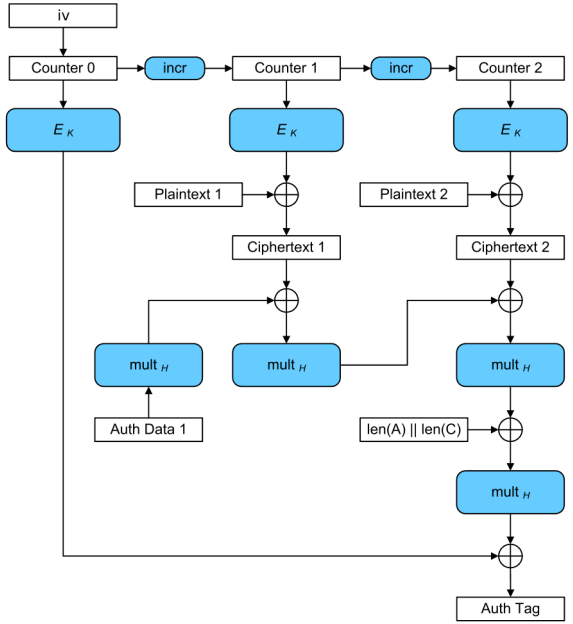
- The Android client can control the salt -> key reuse

IV Reuse Attack (v15/v20-s9)

- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse

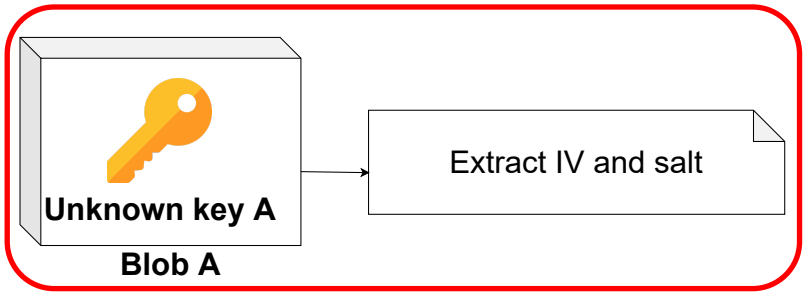
IV Reuse Attack (v15/v20-s9)

- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



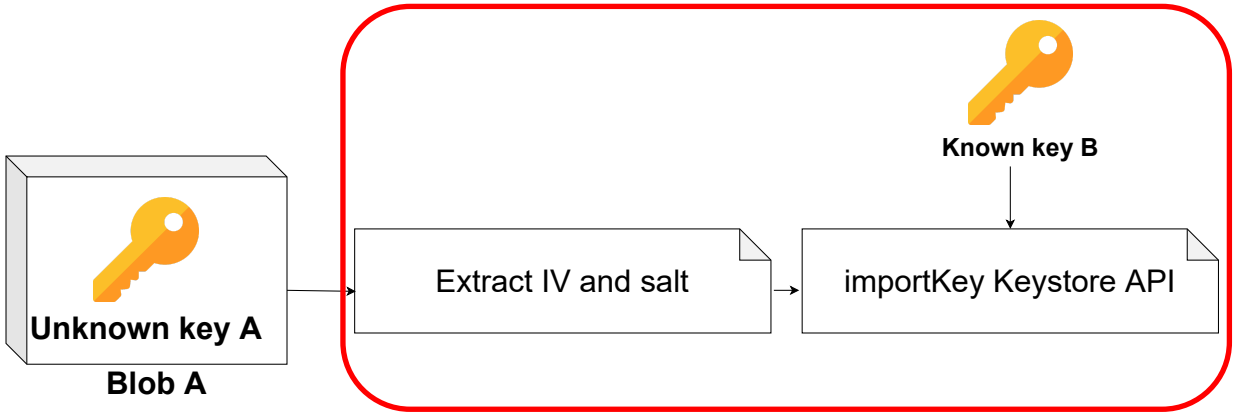
IV Reuse Attack (v15/v20-s9)

- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



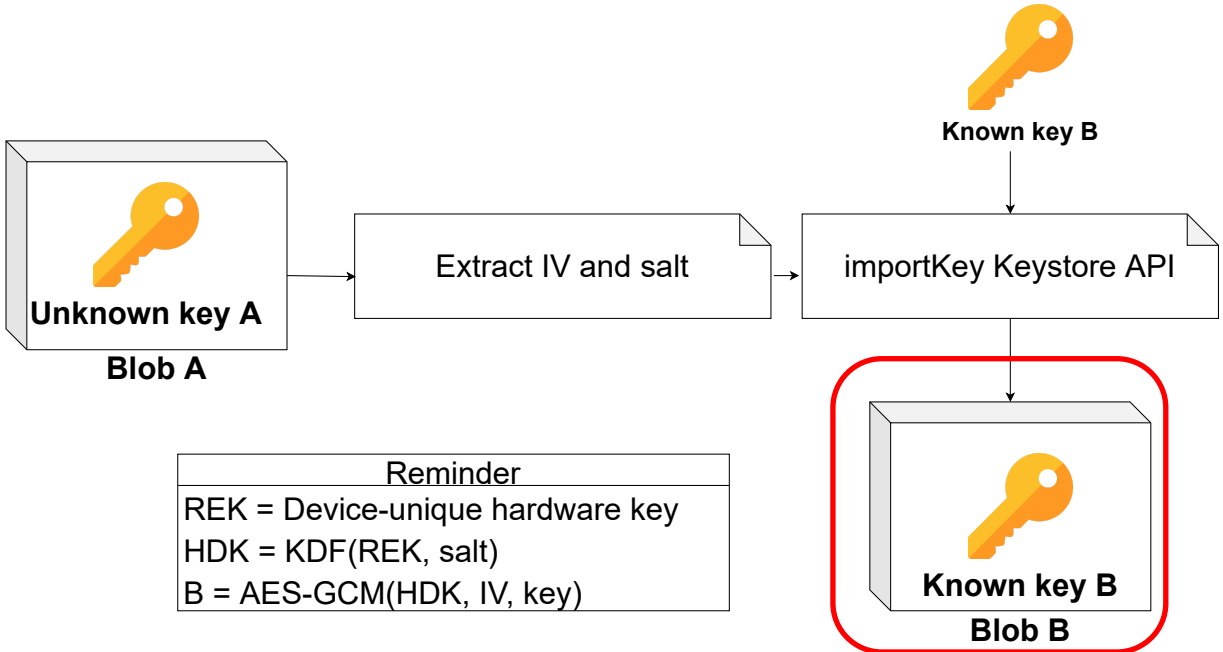
IV Reuse Attack (v15/v20-s9)

- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



IV Reuse Attack (v15/v20-s9)

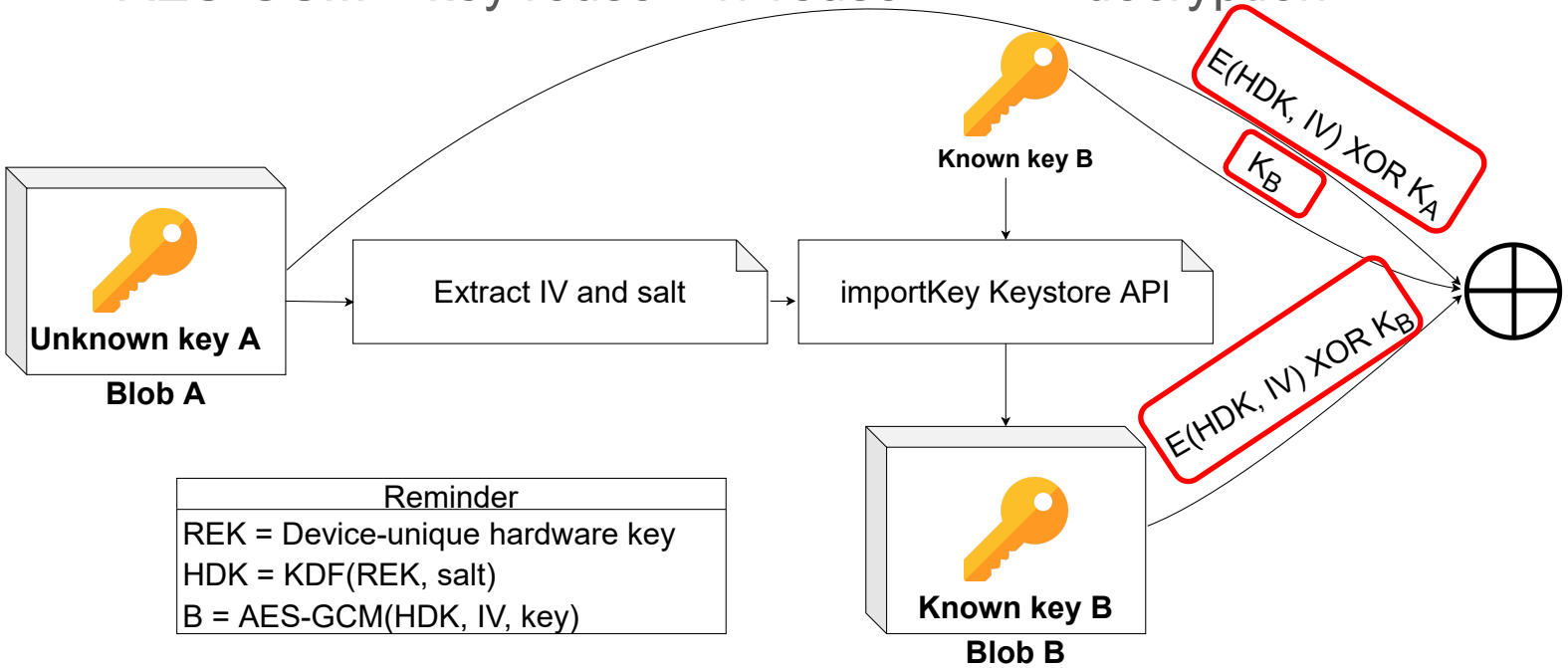
- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



Reminder
REK = Device-unique hardware key
HDK = KDF(REK, salt)
B = AES-GCM(HDK, IV, key)

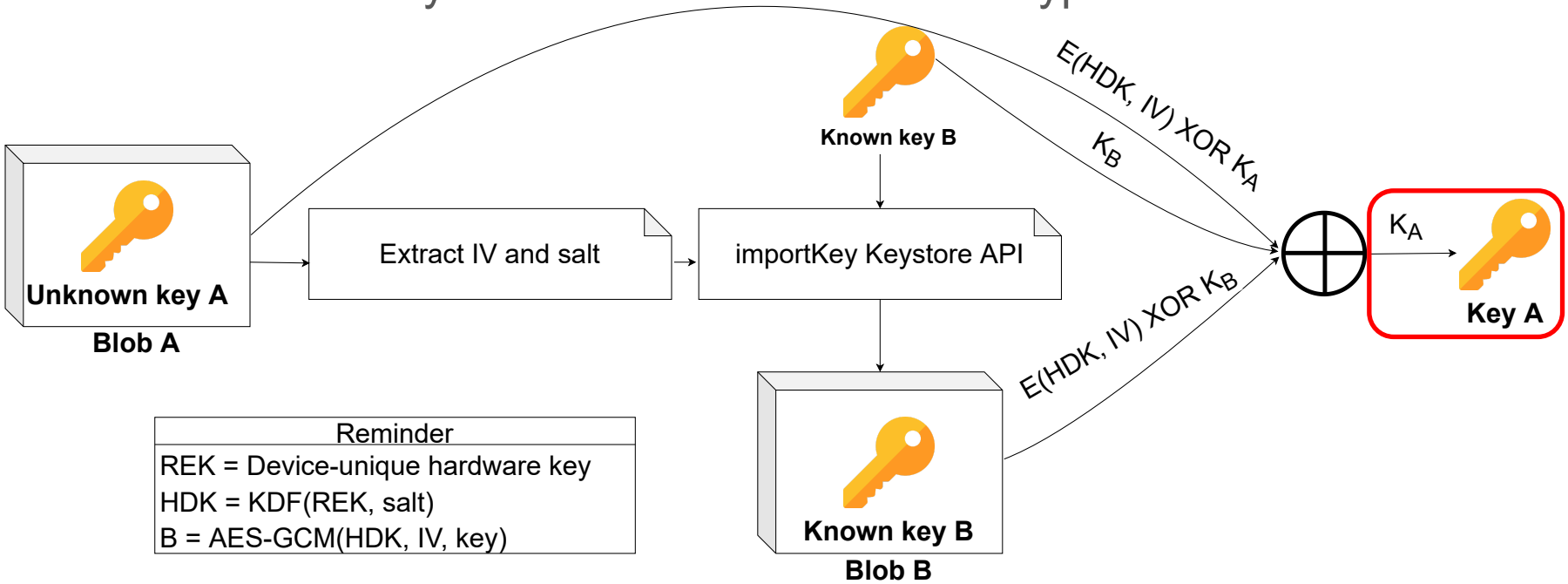
IV Reuse Attack (v15/v20-s9)

- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



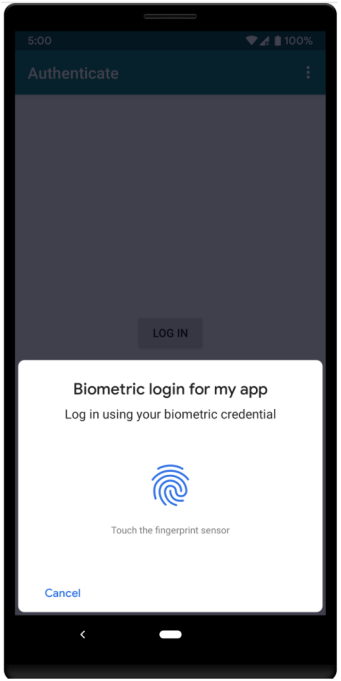
IV Reuse Attack (v15/v20-s9)

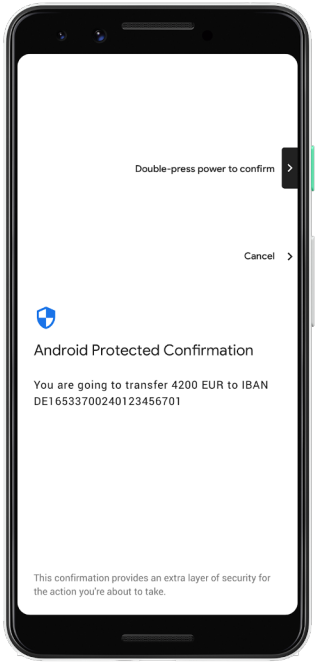
- The Android client can control the salt -> key reuse
- The Android client can control the IV -> IV reuse
- AES-GCM + key reuse + iv reuse -> decryption



Bypassing Authentication and Confirmation

We can bypass any key usage restriction without user presence/consent

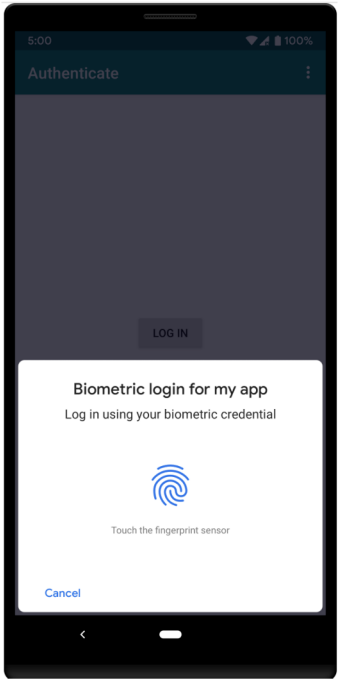


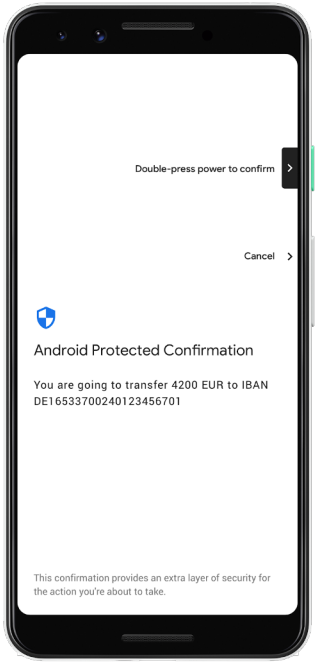


Images from Android Developers Blog

Bypassing Authentication and Confirmation

We can bypass any key usage restriction without user presence/consent

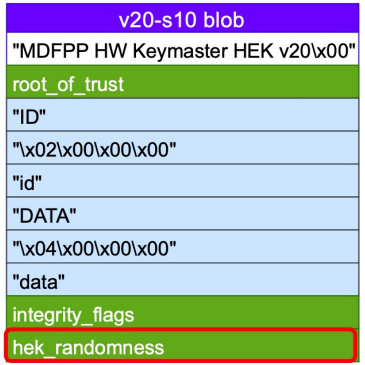




Images from Android Developers Blog

Downgrade Attack

- V20-s10 has randomized salt –> no trivial key reuse



Downgrade Attack

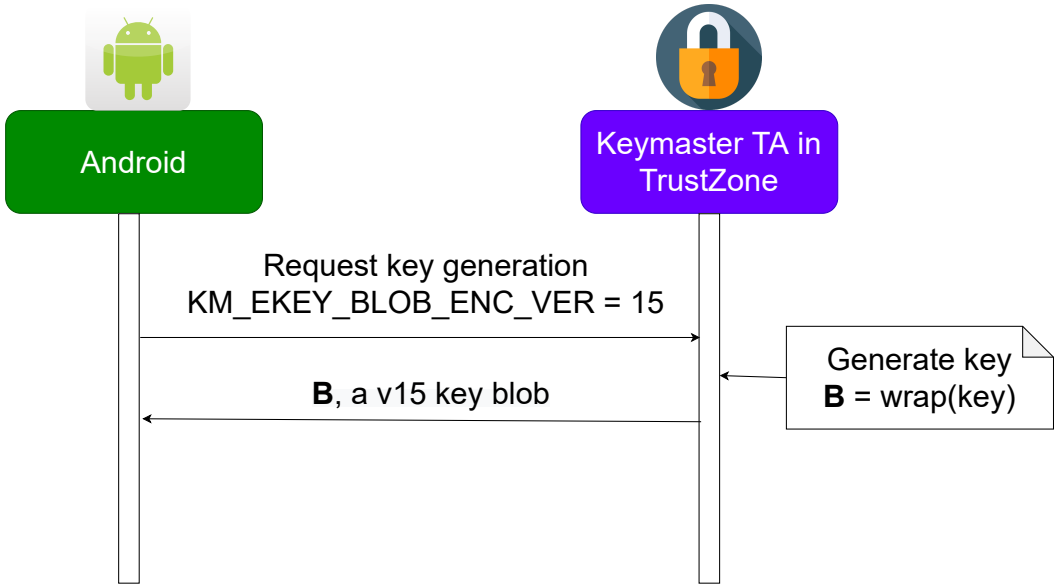
- V20-s10 has randomized salt –> no trivial key reuse
- **Latent code** allows creation of v15 blobs

v20-s10 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags
hek_randomness

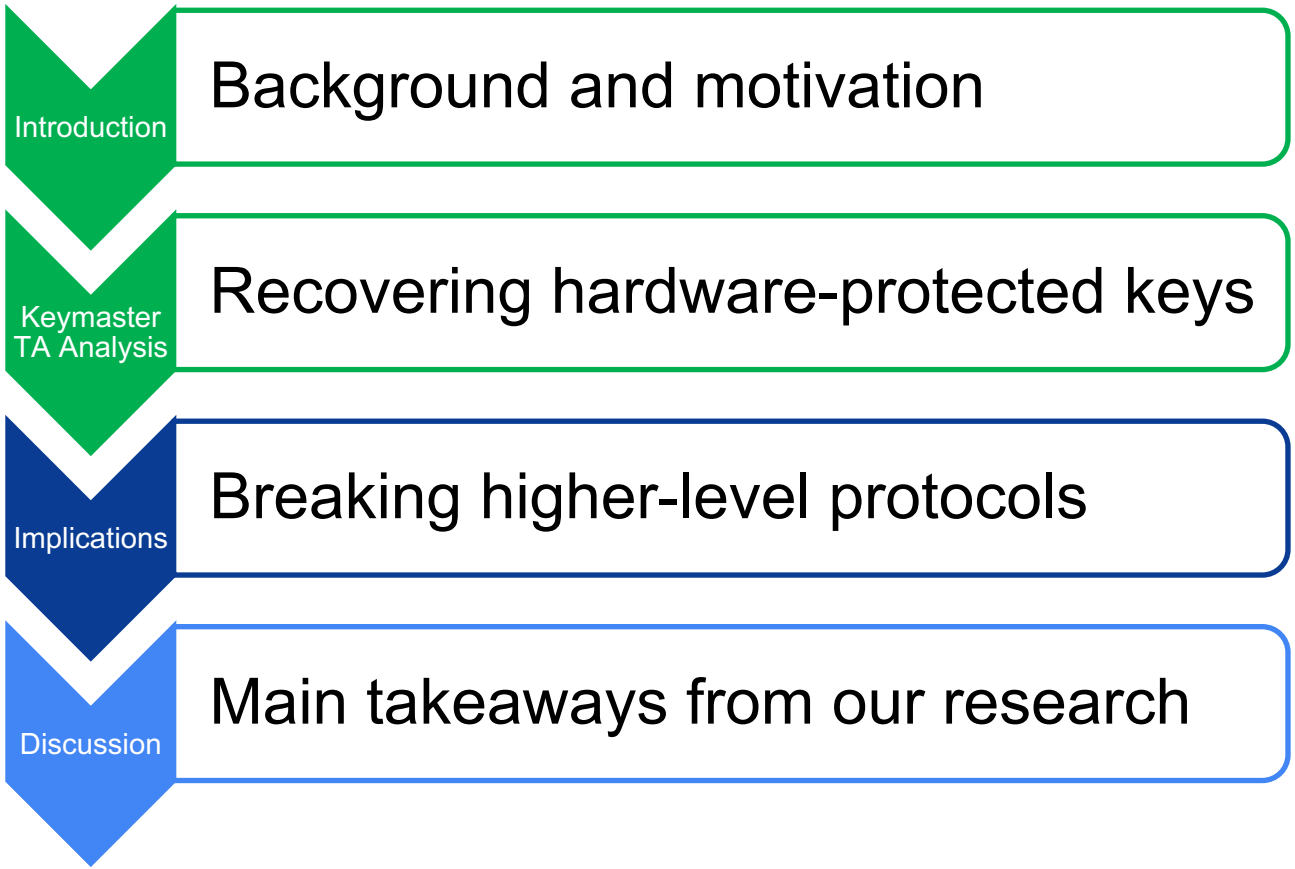
Downgrade Attack

- V20-s10 has randomized salt –> no trivial key reuse
- **Latent code** allows creation of v15 blobs
- A privileged attacker can exploit this to force all new blobs to version v15

v20-s10 blob
"MDFPP HW Keymaster HEK v20\x00"
root_of_trust
"ID"
"\x02\x00\x00\x00"
"id"
"DATA"
"\x04\x00\x00\x00"
"data"
integrity_flags
hek_randomness



Agenda


Introduction
Background and motivation
Keymaster
TA Analysis
Recovering hardware-protected keys
Implications
Breaking higher-level protocols
Discussion
Main takeaways from our research

FIDO2 WebAuthn

Allows passwordless authentication

FIDO2 WebAuthn

Allows passwordless authentication

Authentication keys live inside a “platform authenticator”

FIDO2 WebAuthn

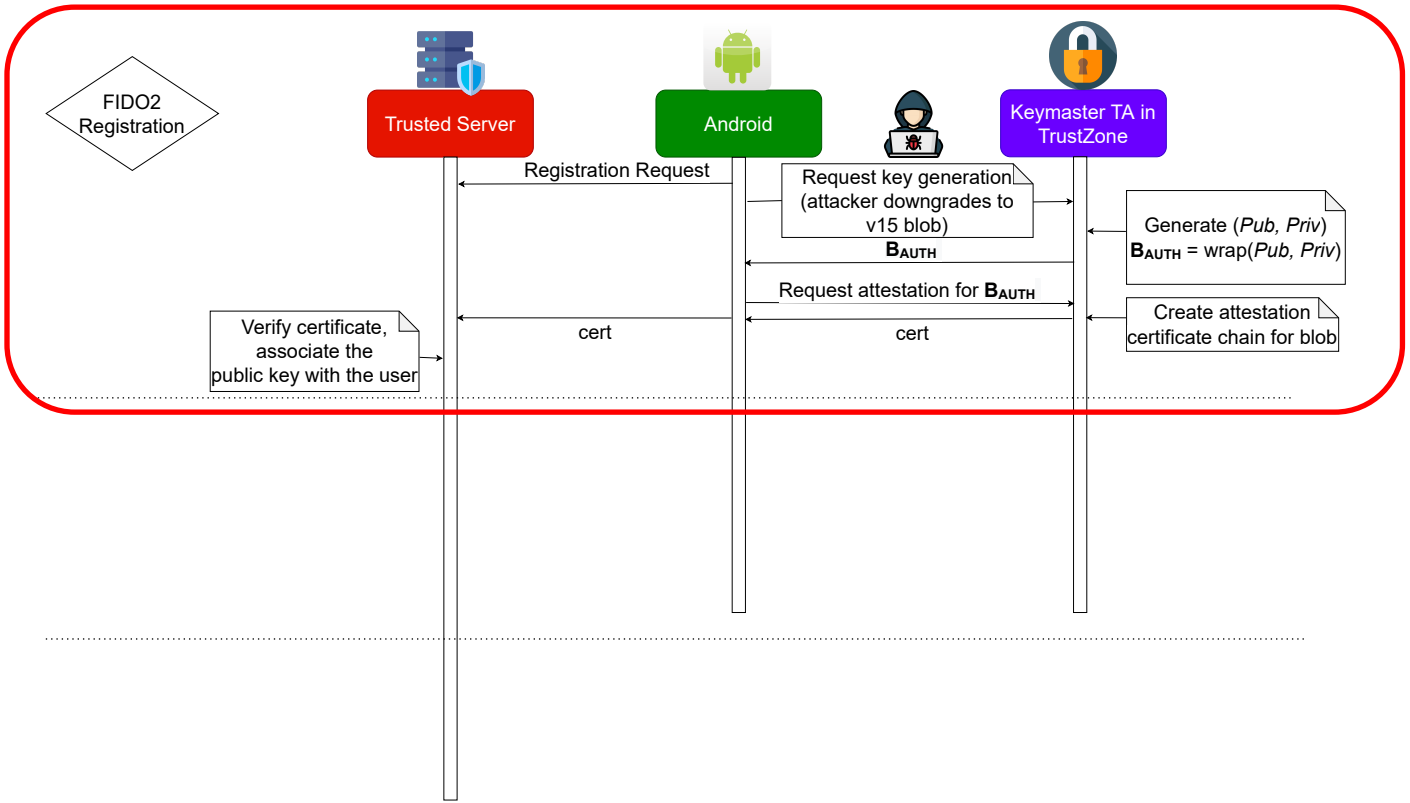
Allows passwordless authentication

Authentication keys live inside a “platform authenticator”

Hard to extract the keys from the secure element

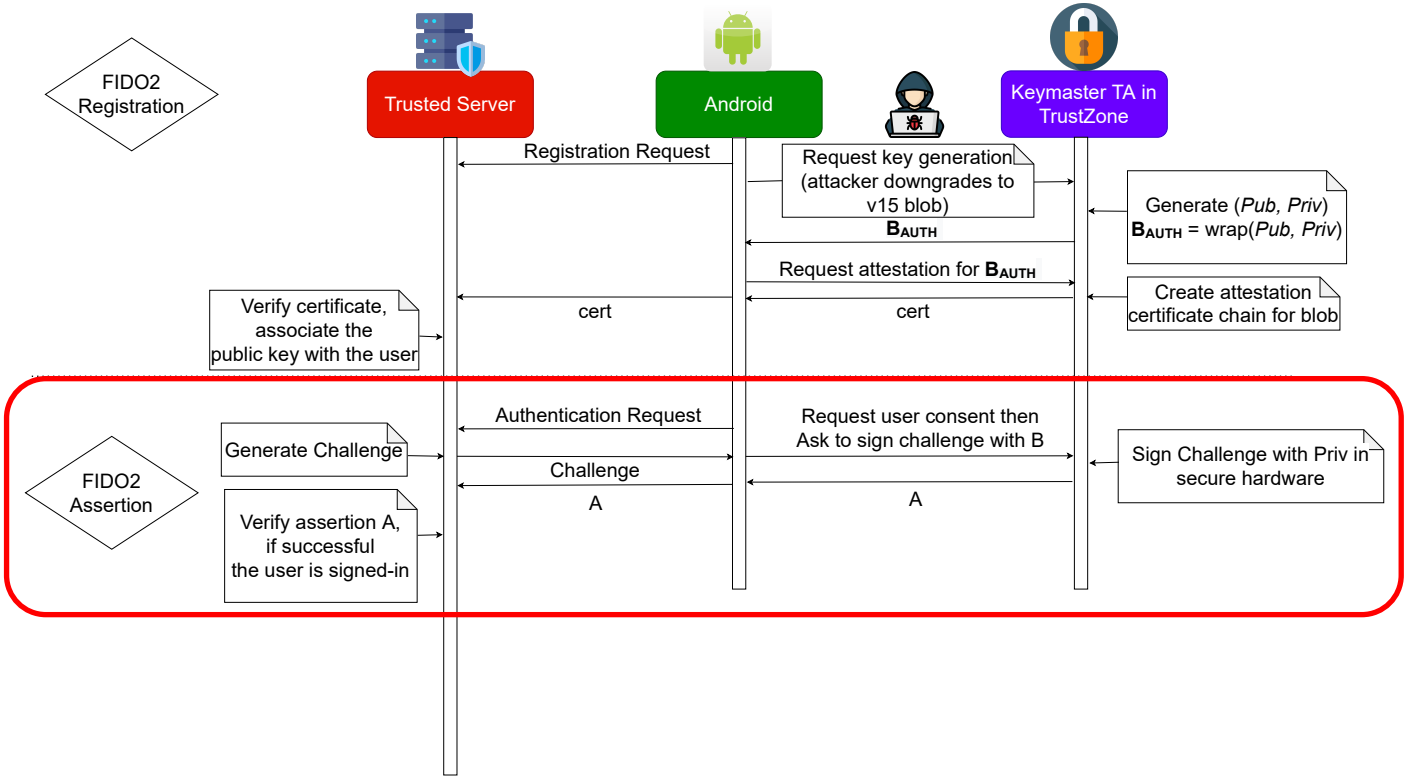
Or to clone the platform authenticator

Bypassing FIDO2 WebAuthn



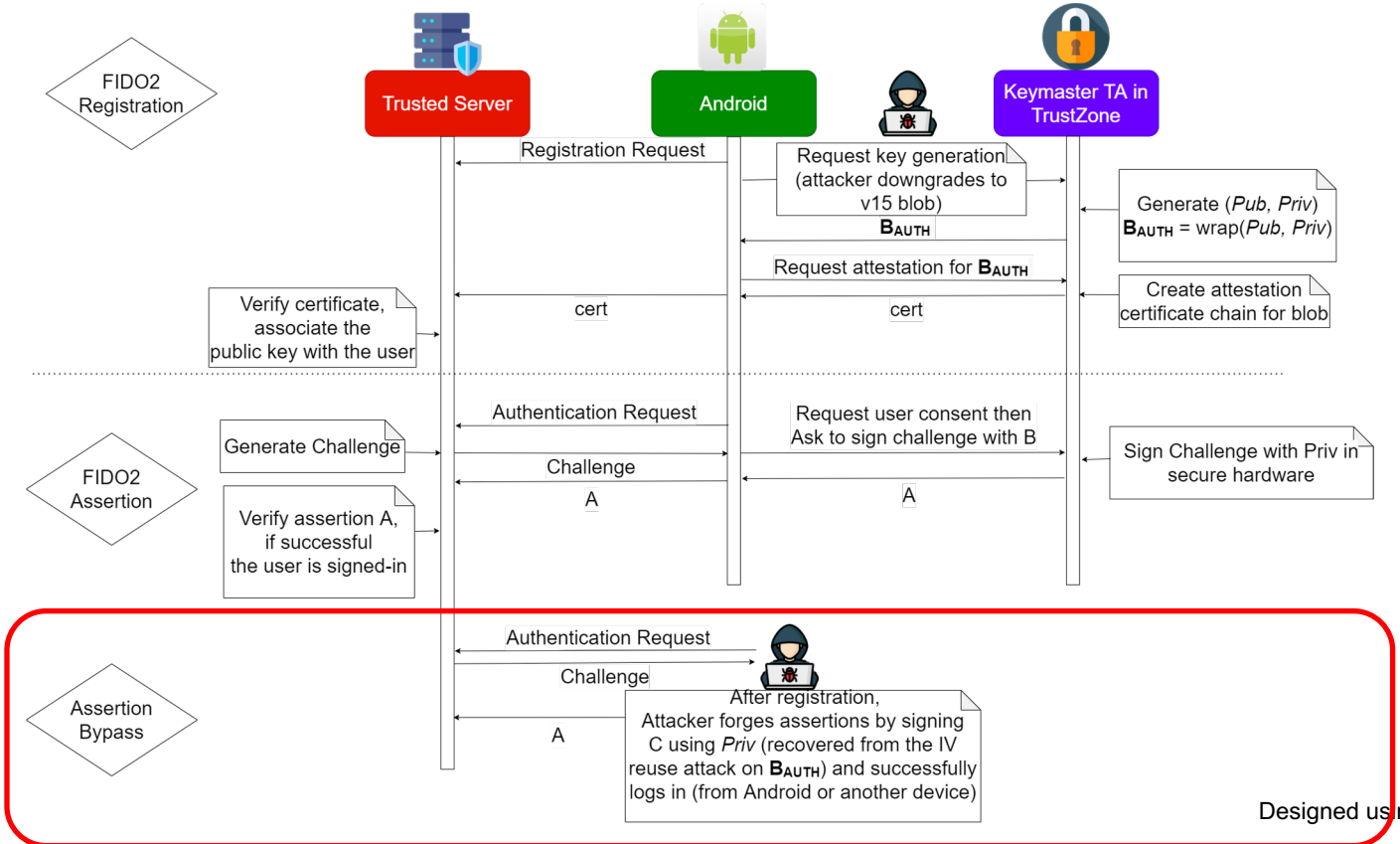
Designed using resources from Flaticon.com

Bypassing FIDO2 WebAuthn



Designed using resources from Flaticon.com

Bypassing FIDO2 WebAuthn



Designed using resources from Flaticon.com

Bypassing FIDO2 WebAuthn Demo #1

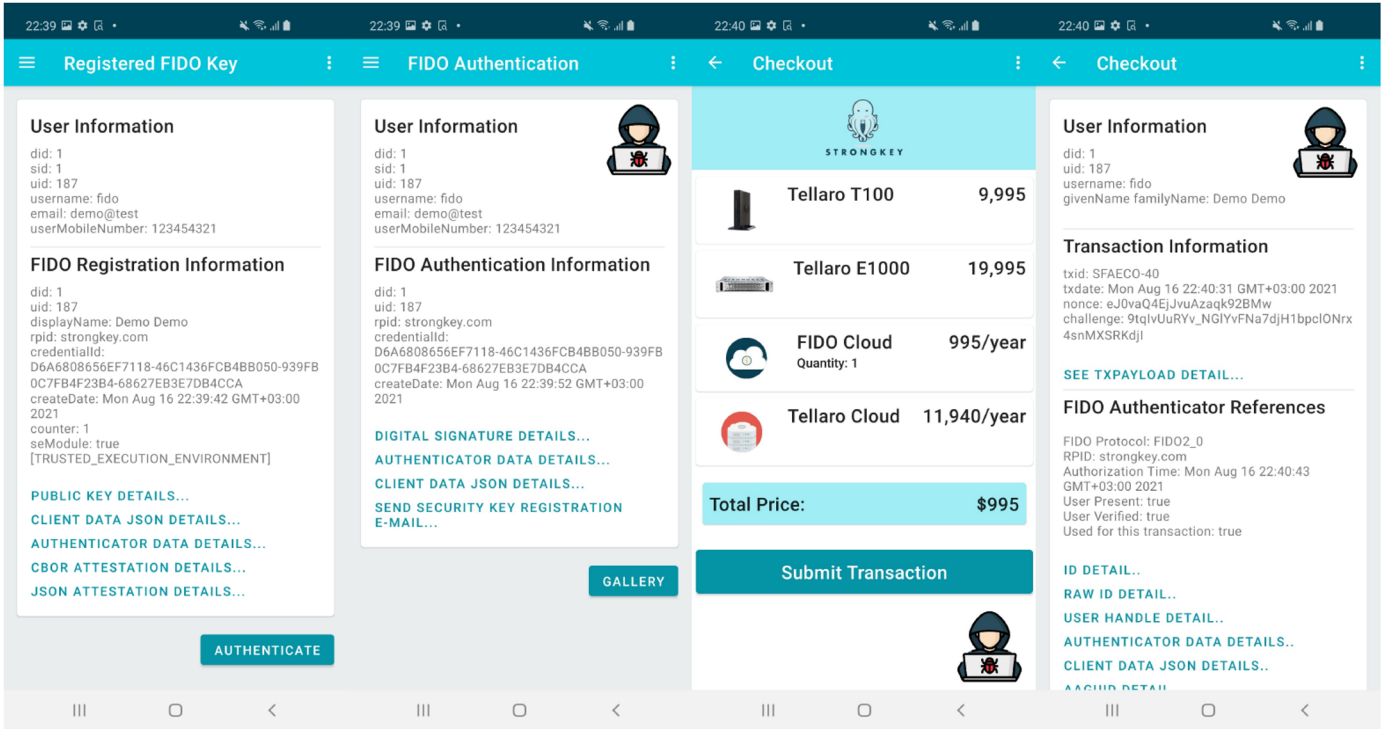
```
beyond1:/data/local/tmp # ./gdbserver --attach :1337 $(pidof android.hardware.keymaster@4.0-service)
Attached; pid = 5190
Listening on port 1337
```

(a) Attaching a GDB debugger to the Keymaster HAL process

```
Breakpoint 2, 0x00000077dae6e514 in nwd_generate_key () from target:/vendor/lib64/libkeymaster_helper_vendor.so
intercepted request to nwd_generate_key
copy old key parameters to new buffer
$1 = 0x7759c24000
$2 = 0x7759c24000
add new parameter (KM_EKEY_BLOB_ENC_VER, 15)
switch to new parameters - this forces the generation of a v15 blob
Breakpoint 4, 0x00000077dae6e544 in nwd_generate_key () from target:/vendor/lib64/libkeymaster_helper_vendor.so
dump the key blob that the keymaster returned
start 0x7759c3b280, end 0x7759c3b4d2, len 252
dumped to result.bin
```

(b) During registration, the GDB script performs the downgrade attack

Bypassing FIDO2 WebAuthn Demo #2



(c) Registration success (d) Authentication success (e) Checkout example (f) Re-authentication success

What did we find?

Attackers could steal cryptographic keys of applications

Attackers could steal your identity

Responsible Disclosure #1

- May ‘21: We reported the IV reuse attack on S9 to Samsung

Responsible Disclosure #1

- May ‘21: We reported the IV reuse attack on S9 to Samsung
- Aug ‘21: Samsung patched Android O/P/Q devices
  - S9, J3 Top, J7 Top, J7 Duo, TabS4, Tab-A-S-Lite, A6 Plus, A9S
  - CVE-2021-25444 with High severity
  - Removed the option to add a custom IV from the API

SVE-2021-21948 (CVE-2021-25444): IV reuse in Keymaster TA

Severity: High
Affected versions: O(8.1), P(9.0), Q(10.0)
Reported on: May 25, 2021
Disclosure status: Privately disclosed.
An IV reuse vulnerability in keymaster prior to SMR AUG-2021 Release 1 allows decryption of custom keyblob with privileged process.
The patch prevents reusing IV by blocking addition of custom IV.

Responsible Disclosure #1

- May ‘21: We reported the IV reuse attack on S9 to Samsung
- Aug ‘21: Samsung patched Android O/P/Q devices
  - S9, J3 Top, J7 Top, J7 Duo, TabS4, Tab-A-S-Lite, A6 Plus, A9S
  - CVE-2021-25444 with High severity
  - Removed the option to add a custom IV from the API

SVE-2021-21948 (CVE-2021-25444): IV reuse in Keymaster TA

Severity: High
Affected versions: O(8.1), P(9.0), Q(10.0)
Reported on: May 25, 2021
Disclosure status: Privately disclosed.
An IV reuse vulnerability in keymaster prior to SMR AUG-2021 Release 1 allows decryption of custom keyblob with privileged process.
The patch prevents reusing IV by blocking addition of custom IV.

Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
  - “There is no application created with the key blob version as v15. And any of the applications cannot change its key blob version for it to be exploitable.”

Responsible Disclosure #2

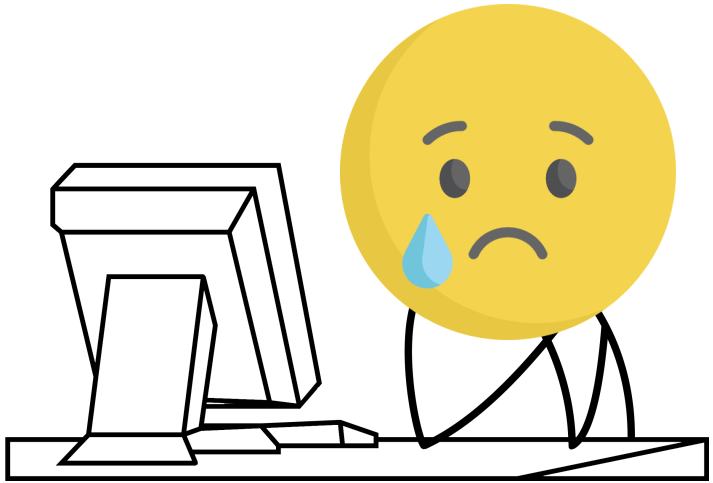
- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21

Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
  - “we think that there is no practical security impact on this”

Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
  - “we think that there is no practical security impact on this”



Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
- Aug ‘21: We sent the paper

Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
- Aug ‘21: We sent the paper
- Sep ‘21: Samsung reviewed and re-investigated the impact
  - “After further review of your paper, we concluded that "Downgrade Attack" also has practical impact with our devices”

Responsible Disclosure #2

- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
- Aug ‘21: We sent the paper
- Sep ‘21: Samsung reviewed and re-investigated the impact
  - “After further review of your paper, we concluded that "Downgrade Attack" also has practical impact with our devices”

Responsible Disclosure #2

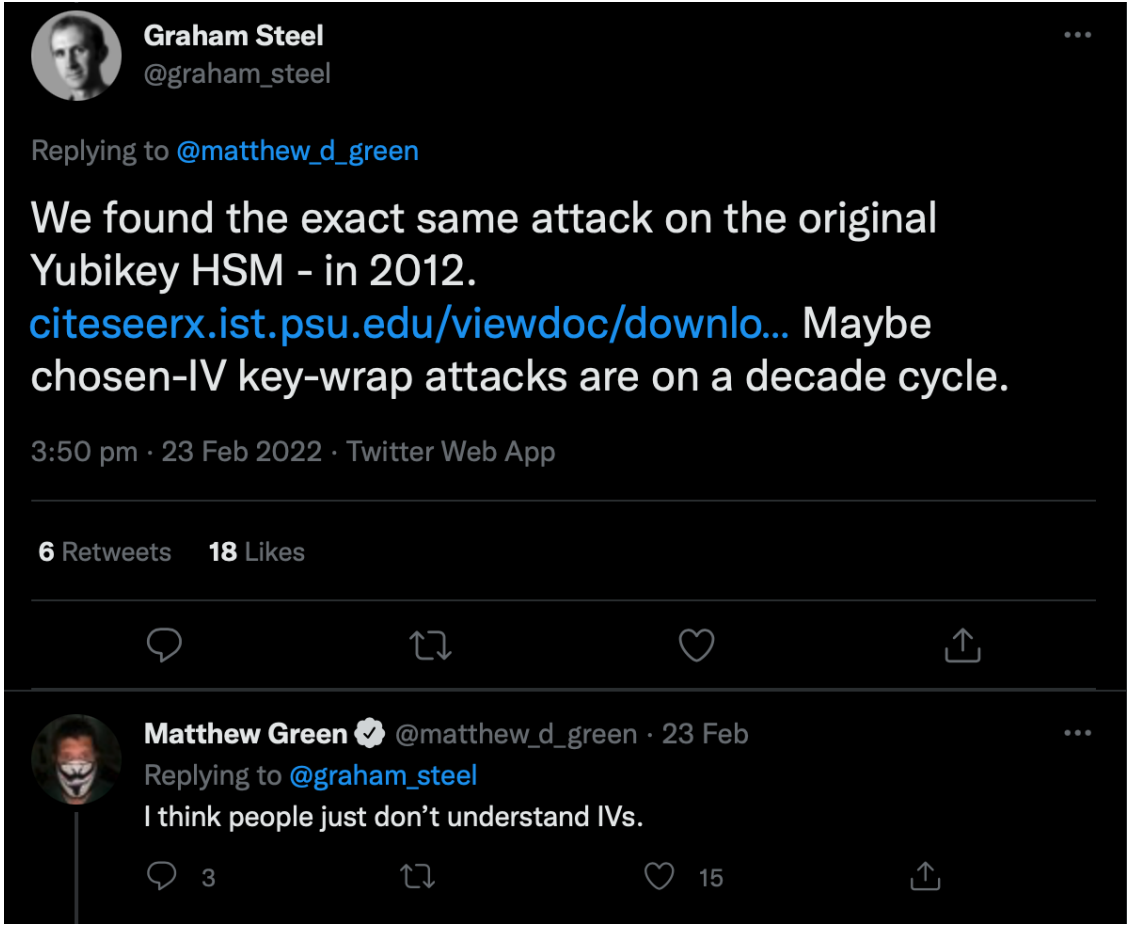
- Jun ‘21: Samsung rejected the downgrade attack
- Jul ‘21: We reported the downgrade attack on S10, S20 and S21
- Aug ‘21: Samsung rated the downgrade attack as “very Low severity”
- Aug ‘21: We sent the paper
- Sep ‘21: Samsung reviewed and re-investigated the impact
- Oct ‘21: Samsung patched Android P or later, including S10/S20/S21
  - CVE-2021-25490 with High severity
  - Released a patch that completely removes the legacy key blob implementation

SVE-2021-22658 (CVE-2021-25490): Downgrade attack in Keymaster TA

Severity: High
Affected versions: P(9.0), Q(10.0), R(11.0)
Reported on: July 16, 2021
Disclosure status: Privately disclosed.
A keyblob downgrade attack in keymaster prior to SMR Oct-2021 Release 1 allows attacker to trigger IV reuse vulnerability with privileged process.
The patch removes the legacy implementation for minor keyblob.

No Security By Obscurity

Return of the IV Reuse Attack



Graham Steel
@graham_steel

Replying to @matthew_d_green

We found the exact same attack on the original Yubikey HSM - in 2012. citeseerx.ist.psu.edu/viewdoc/downlo... Maybe chosen-IV key-wrap attacks are on a decade cycle.

3:50 pm · 23 Feb 2022 · Twitter Web App

6 Retweets **18** Likes

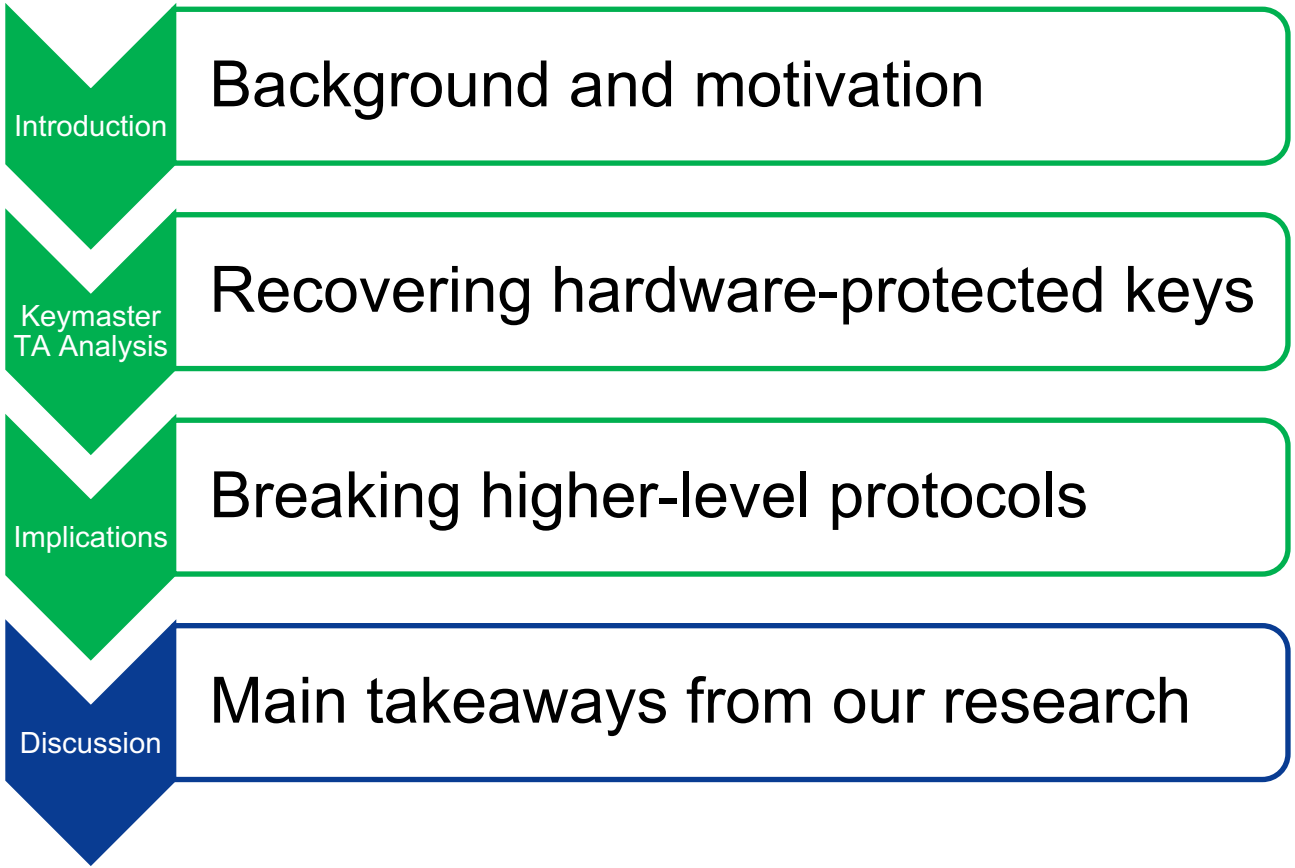
Matthew Green @matthew_d_green · 23 Feb

Replying to @graham_steel

I think people just don't understand IVs.

3 15

Agenda


Introduction
Background and motivation
Keymaster TA Analysis
Recovering hardware-protected keys
Implications
Breaking higher-level protocols
Discussion
Main takeaways from our research

Low-Level Cryptographic Issues

- ❌ Allowing client to set IV
- ❌ Allowing client to set encryption version
- ❌ Latent code in security-critical application
- ❌ Encryption version persists across “upgrades”

Low-Level Cryptographic Issues

❌ Allowing client to set IV
❌ Allowing client to set encryption version
❌ Latent code in security-critical application
❌ Encryption version persists across “upgrades”

✅ Use a unique IV / misuse resistant AEAD (AES-GCM-SIV) / Tink
✅ Disallow choice of encryption version
✅ Reduce attack surface in security-critical application
✅ Always use the latest encryption version

The Gap in Composability

- ❌ Key attestation does not commit to the cryptographic method
- ❌ Closed vendor-specific implementation

The Gap in Composability

- ❌ Key attestation does not commit to the cryptographic method
- ❌ Closed vendor-specific implementation

- ✅ Include encryption version in attestation certificate
- ✅ Uniform open-standard by Google for the Keymaster HAL and TA

Upgrading Android Attestation: Remote Provisioning

25 March 2022

Posted by Max Bires, Software Engineer

Why Change?

The two primary motivating factors for changing the way we provision attestation certificates to devices are to allow devices to be recovered post-compromise and to tighten up the attestation supply chain. In today's attestation scheme, if a device model is found to be compromised in a way that affects the trust signal of an attestation, or if a key is leaked through some mechanism, the key must be revoked. Due to the increasing number of services that rely on the attestation key signal, this can have a large impact on the consumer whose device is affected.

This change allows us to stop provisioning to devices that are on known-compromised software, and remove the potential for unintentional key leakage. This will go a long way in reducing the potential for service disruption to the user.

Image from Android Developers Blog

Conclusions

Fragmented blackbox designs -> dangerous pitfalls

Open standard design

Conclusions

Fragmented blackbox designs -> dangerous pitfalls

Open standard design

No Security By Obscurity

Formal analysis by independent researchers

Conclusions

Fragmented blackbox designs -> dangerous pitfalls

Open standard design

No Security By Obscurity

Formal analysis by independent researchers

Decades of IV reuse in AES-GCM

Misuse-resistant AEAD / cryptography library

Any questions?

- Extended paper: https://eprint.iacr.org/2022/208.pdf
- Tool + PoC: https://github.com/shakevsky/keybuster

@shakevsky

@eyalr0

yash@eng.tau.ac.il