Evolution of Android Keystore

Gautam Arvind Pandian Vikas Gupta

September 24, 2021

Thales DIS

BSides Singapore 2021

Intro

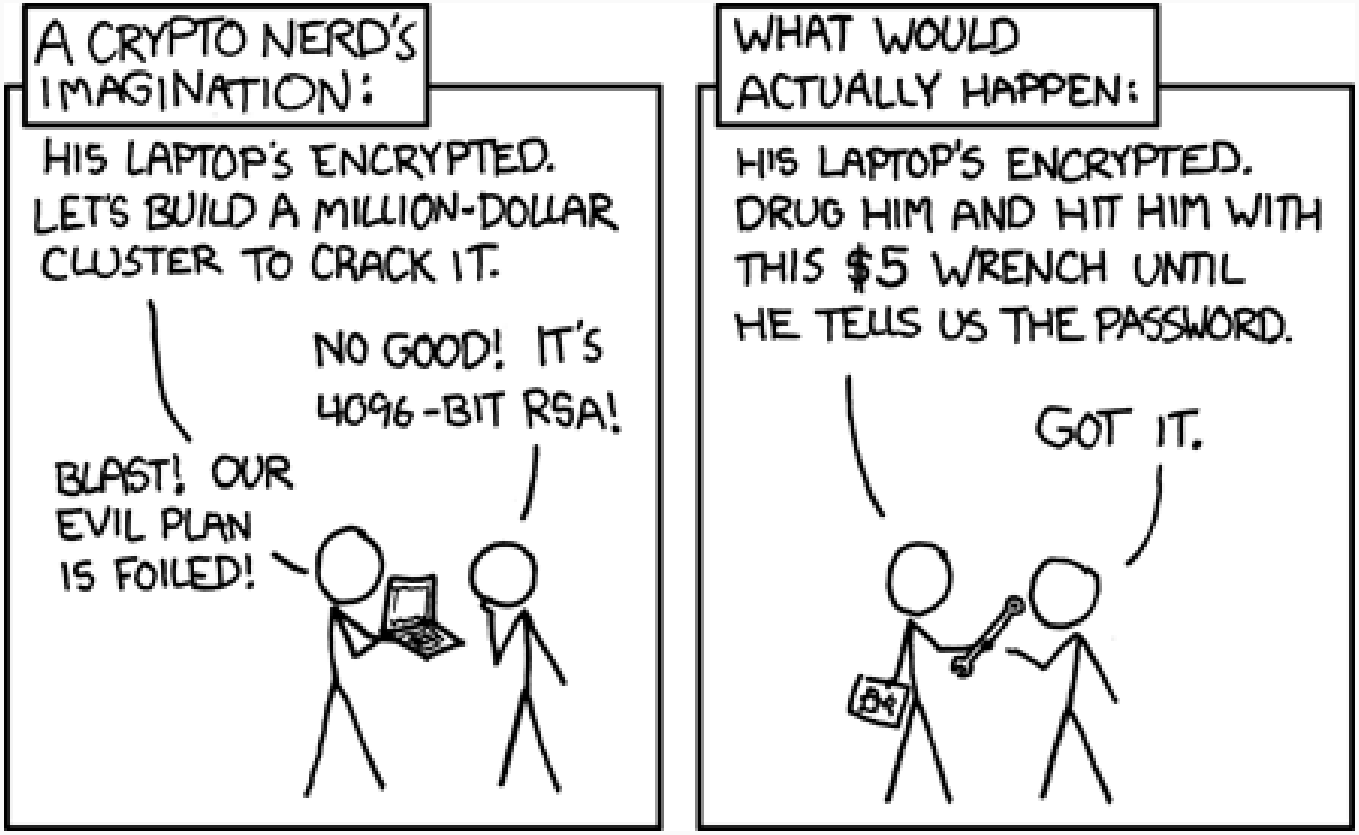
About Us

- **Gautam Arvind Pandian**
  - Security Researcher and Architect at *Thales DIS*
  - CTF creator in r2con2020 conference - R2Pay.
  - Speaker at various conferences - *Android Security Symposium 2020*, *Sincon 2020*
  - Interests: Crypto, Hardening Mobile Apps
  - Github: @darvincisec

- **Vikas Gupta**
  - Security Researcher and Pentester at *Thales DIS*
  - M.Sc in Information Security, OSCP certified
  - Co-Author contributor for *OWASP Mobile Security Testing Guide (MSTG)*
  - Interests: Reverse Engineering, Obfuscation, Crypto
  - Github: @su-vikas

Overview

- What is a Keystore?
- Android Keystore features
  - Credential Store
  - Key Attestation
  - Secure key import
- Attacking Keystore
  - Breaking the app binding
  - The missing attestation
- Comparison with iOS Keychain
- Improvements
- Takeaway



- Law #7: *Encrypted data is only as secure as the decryption key.* (Ten Immutable Laws Of Security v2.0)

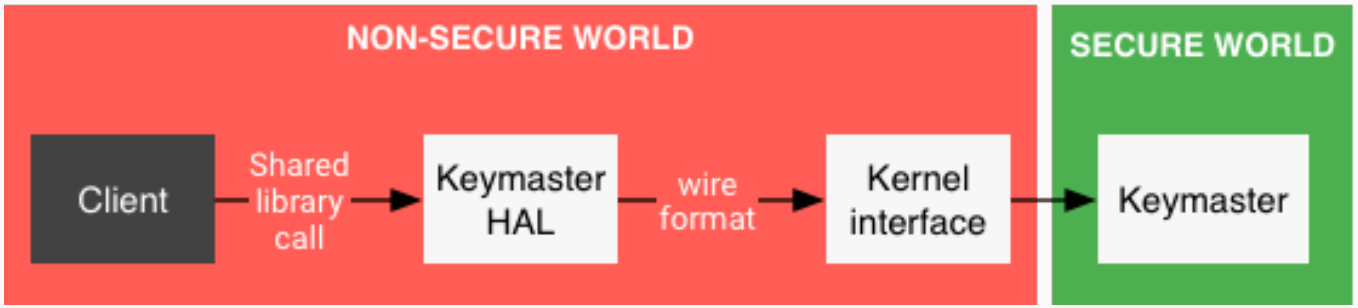
What is a Keystore?

What is a Keystore?

- Repository to store cryptographic keys and other secrets.
- Protects integrity and confidentiality of the stored crypto material.
- Can be a software file or hardware backed.
  - ARM Trustzone
  - Secure Element – eg. Strongbox
- Items that can be stored:
  - Symmetric keys
  - Asymmetric key pairs / Certificates

Architecture of Android Keystore

- Components
  - Android Keystore APIs
  - Keymaster HAL
- Android Keystore API and Keymaster HAL provides basic crypto primitives
  - access-control
  - hardware-backed keys
- Keymaster HAL is OEM-provided, a dynamic loadable library.

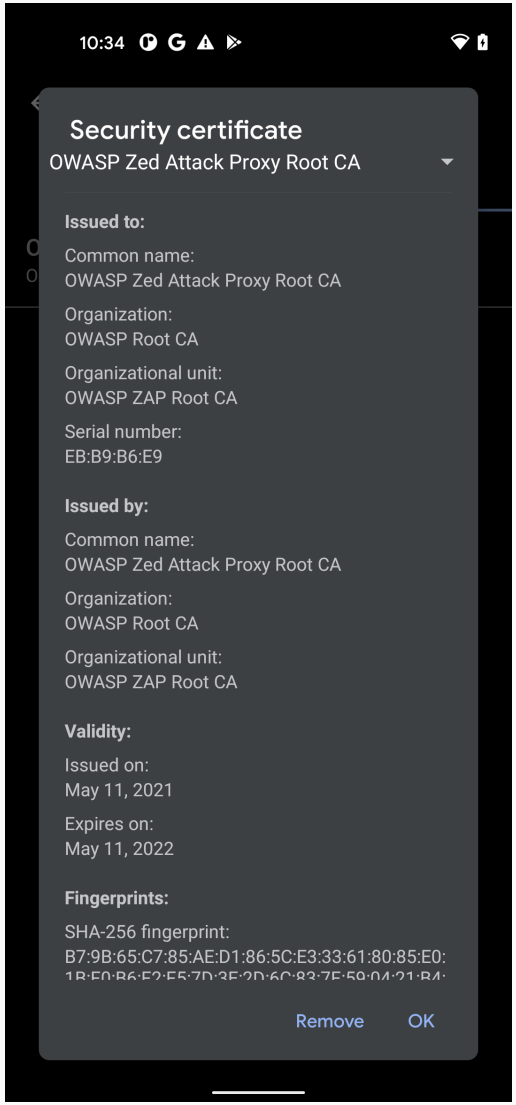


Keystore Before KitKat

Credential Store

- How to store WiFi-EAP and VPN credentials?
- Credential store was added in Android 1.6.
  - Software based solution.
  - Available only to system application – Settings, WiFi and VPN
  - Not available for 3rd party usage.
- Stored credentials available system-wide.

Credential Store



Credential Store

- No 3rd party access to credential store was a barrier for corporate adoption of Android OS.
- Android v4.0 introduced
  - Keychain API: Public API to use the credential store.
  - A better integration with the rest of the OS.
    - Using binder instead of sockets.
  - OS support for hardware-backed secure storage implemented.
- In Android v4.1 Keymaster HAL was added.

Credential Store

- Used software based encryption mechanisms.
- Master key to encrypt stored keys.
  - Master key stored on the filesystem itself.
- Master key derived using
  - 128-bit AES Key using PBKDF2 with 8192 iterations, with 128-bit salt
  - Android pre-4: a dedicated credential store protection password
  - Android 4+: password derived using device unlock PIN, pattern or password

Keystore After Kitkat

Android 6.0

- User authentication was still a problem – weak pin or patterns was common place.
  - Fingerprint based authentication introduced.
  - Fingerprint authentication is bound to Android keystore.
- Keystore can require fingerprint auth before using a stored key.
- Support for AES and HMAC in KeyGenerator added.
- Long history of regression with *setEncryptionRequired* API resolved.

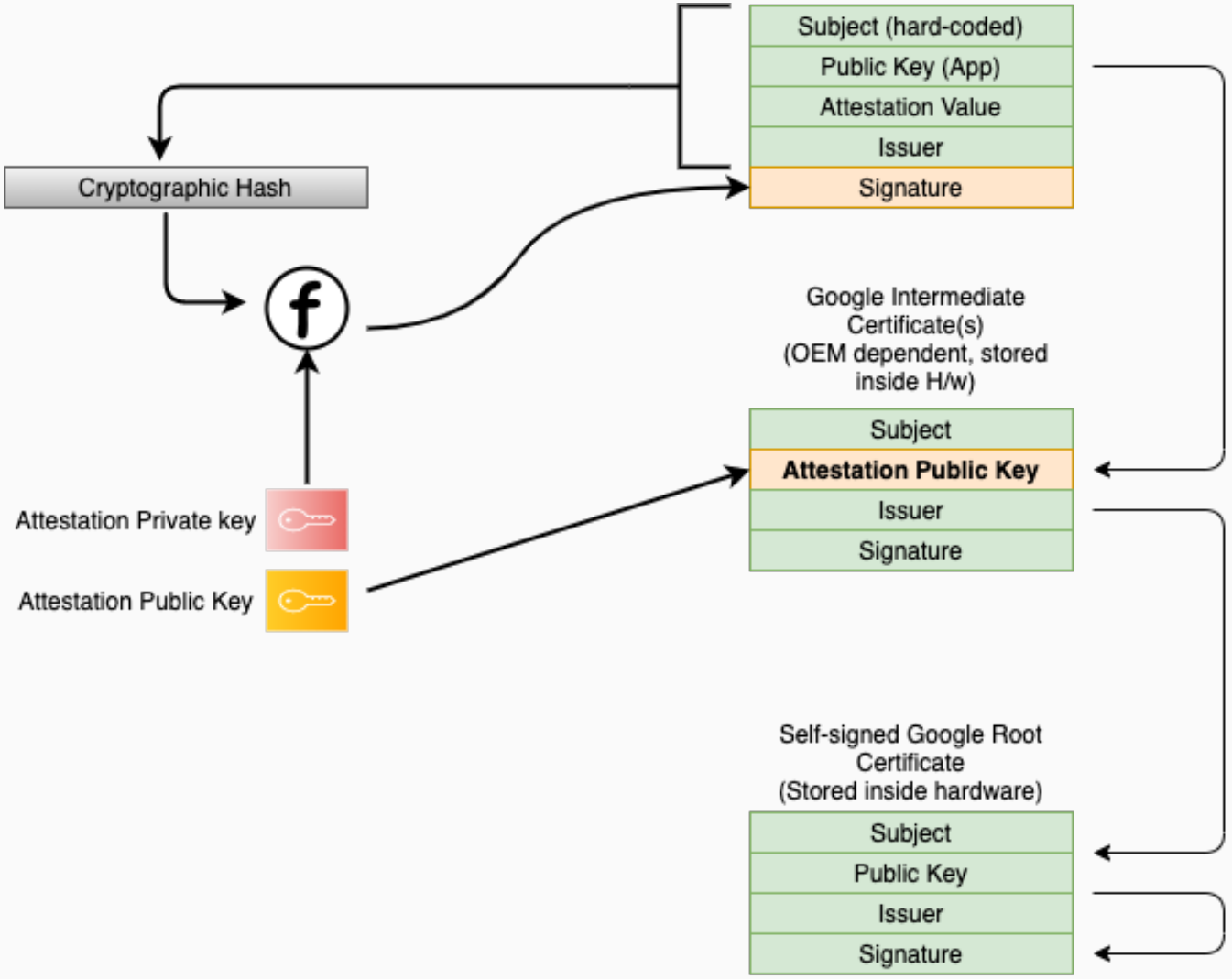
Using Biometric Authentication for Key Access

- Key or Keypair generated/imported in keystore can have the following access control
  - No authentication
  - Key available after device unlock
  - Biometric authentication for every key access
  - Key valid for a period of time after biometric or keyguard authentication
    - Google Pay, Google FIDO2 authentication and contactless payment apps use this option
    - Having weak pin/pattern/password undermines the security of such apps

Key Attestation - Android 7.0

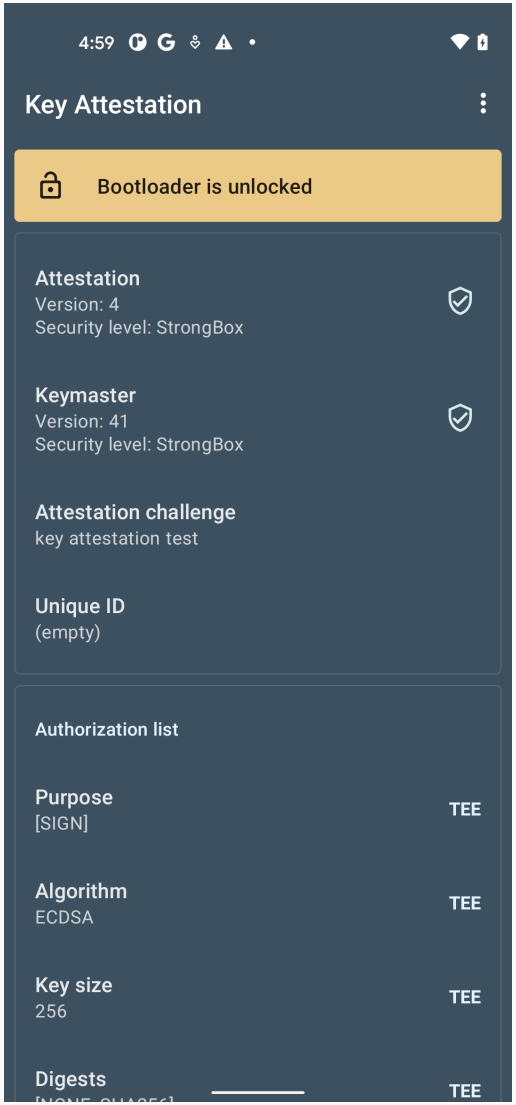
- Q: **How to determine if a given key is in h/w backed keystore or not?**
- Key attestation working
  - Generates a public key certificate that contain a detailed description of the key and its access controls, to make the key's existence in secure hardware and its configuration remotely verifiable.
  - Helps to check the device bootloader locked status, the application id and signature which generated the key pair.

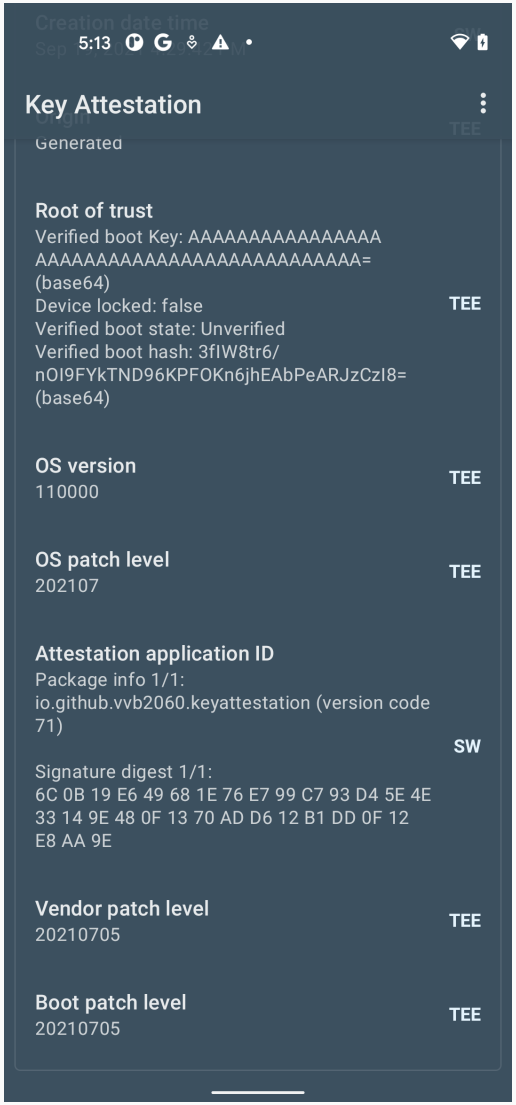
Key Attestation Working



Attestation Chain of Trust

Key Attestation





- Caution: *Some parameters are obtained using software, and can be easily spoofed.*

Key Attestation

- Banks can use to determine if a device is secure enough to store private keys
  - Store private keys for electronic payments.
- FIDO2 specification indicates about including key attestation as part of initial registration operation
  - It allows the service to trust that it knows the provenance of the authenticator being used.
  - Google FIDO2 Authenticator is a well-known FIDO2 Authenticator, but it doesn't use key attestation yet.
- Safetynet attestation internally makes use of Key Attestation to fetch the device integrity status in a non-spoofable way.

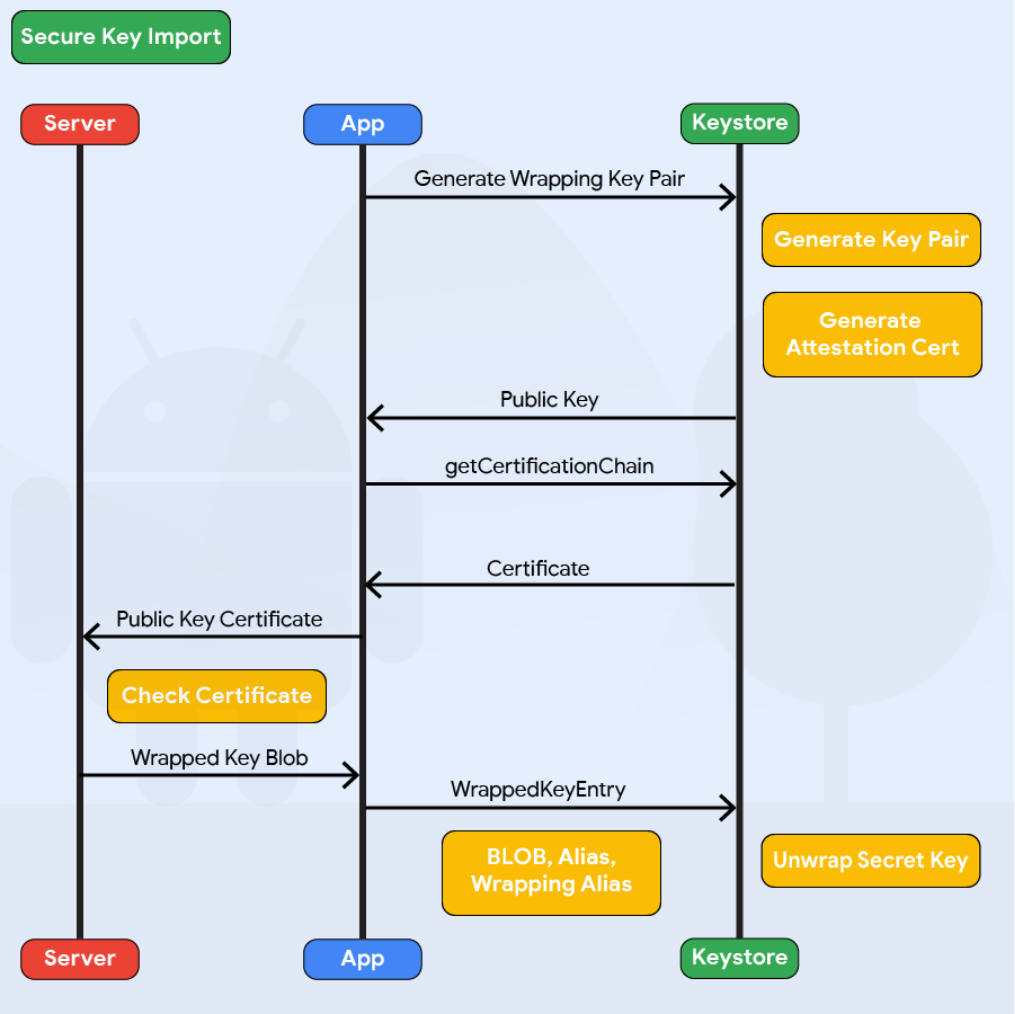
ID Attestation - Android 8.0

- **Q: How to trust an advertised device information?**
- ID attestation is requested by performing a key attestation and including the device identifiers to attest in the request.
  - Brand Name
  - Device Name
  - Product Name
  - Manufacturer Name
  - Model Name
  - Serial Number
  - IMEI
  - MEID
- Further updates in Android 12.

Secure Key Import - Android 9.0

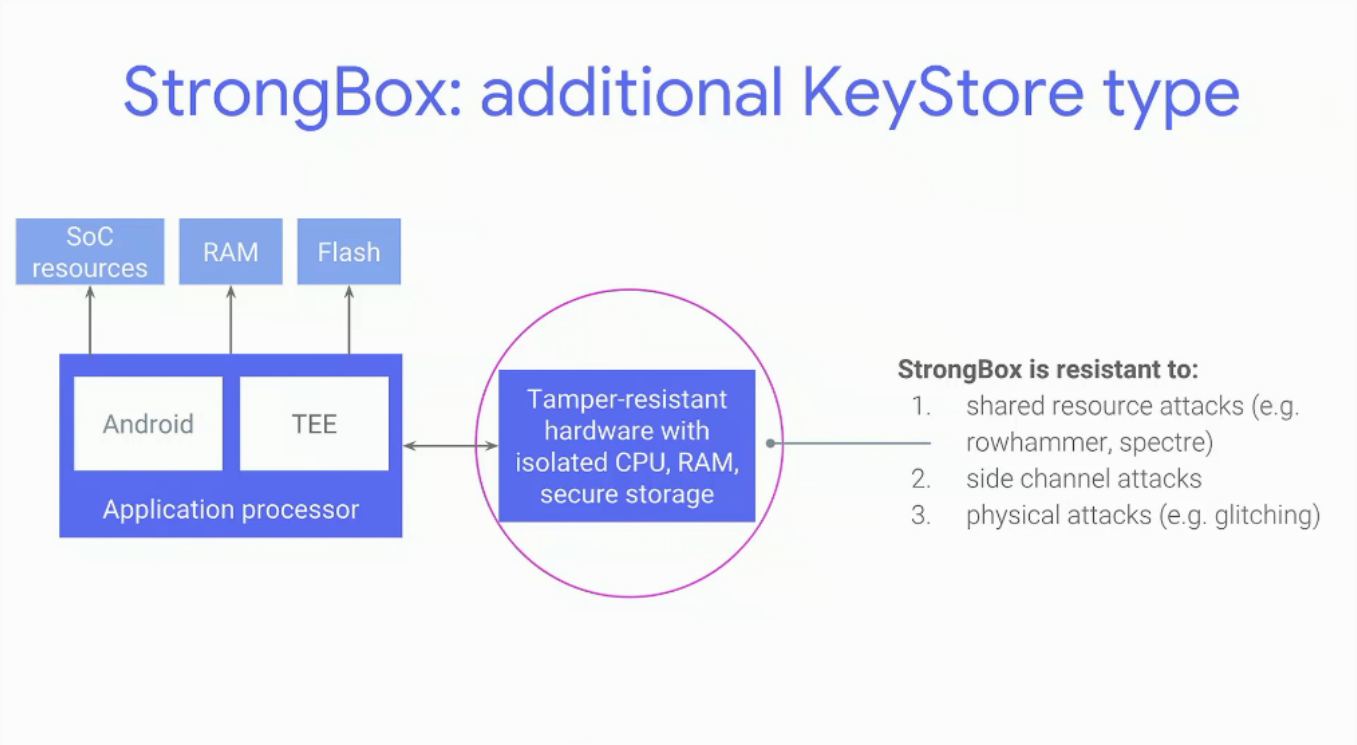
- Provisioning of symmetric keys from server on to device without being exposed in clear in the device memory.
- Server can legitimately verify the request from mobile device is indeed originating from a trusted application executing on a mobile device.
- Google Pay uses Secure Key Import to provision some keys on Pixel 3 phones, to prevent the keys from being intercepted or extracted from memory.

Secure Key Import - Working



Strongbox - Android 9.0

- StrongBox is an implementation of the Keymaster HAL that resides in a hardware security module.



Identity Credential Store - Android 11

- Support for storing identity documents.
- Multiple credential can be stored:
  - Digital keys (car, home, office)
  - Mobile Driver's License (mDL), National ID, ePassports
  - eMoney solutions (for example, Wallet)
- Strongbox and tamper resistant hardware helps to enable this feature.
- Currently apps are performing RASP and other software protections to ensure security and integrity.

Attacks

Threat Model

- Undetectably undermine the security of the applications relying on Keystore.
- **Malicious App Attacker**
  - Attacker tries to attack the secure key storage from another app installed on the same device.
- **Root Attacker**
  - Attacker has root privileges and is able to run apps under root permissions.
  - Attacker can use root exploits
  - Users root their device and some play store apps need root for running - Titanium Backup.
  - Loss or theft of mobile device is very much a plausible scenario

Existing Keystore Protections

- **Device Binding**
  - All keys are protected by encrypting using a hardware backed key, unique to the device.
  - On unlocking device's bootloader, existing crypto material is deleted.
- **Key material never enters the application process memory**
  - All processing either in system process or hardware.
  - Keymaster HAL provides hardware backed cryptographic services.
- **Access control on keys**
  - Only the app that generates/imports the key can access it.
  - Key can be accessed upon authentication.

Existing Keystore Protections

- **Key Authorization**
  - Keys can be assigned a purpose, while generation/importing.
    - Signing
    - Encryption
  - Other properties are also defined while creating it.
    - Algorithm
    - Padding

- **Expiration**
  - Duration of a key's validity

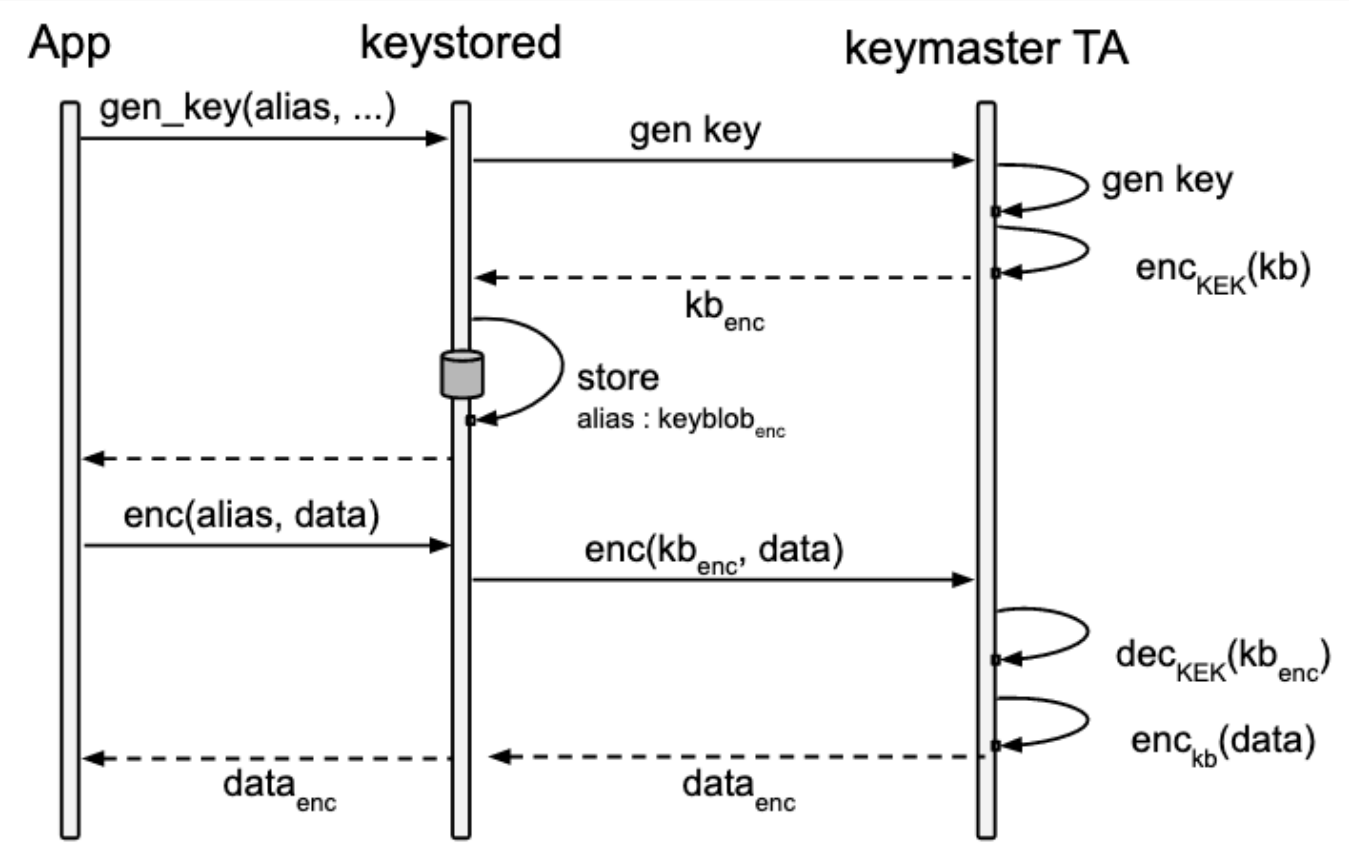
- **Key Attestation**
  - To ensure the keys are stored in hardware backed environment.

Keystore, Keys and KeyBlobs

- Each app gets a unique UID on installation.
  - UID helps in mapping app and its keys.
- A key generated by an app is stored as an encrypted keyblob in */data/misc/keystore/user_x*
  - All the files in this folder are protected with *keystore* SELinux domain.
- The file name indicates UID of the app, key alias, private/symmetric key or certificate.

```
-rw------- 1 keystore keystore    759 2021-09-19 16:29 10268_USRCERT_Key1
-rw------- 1 keystore keystore    748 2021-09-19 16:29 10268_USRCERT_Key2
-rw------- 1 keystore keystore    587 2021-09-19 16:29 10268_USRPKEY_Key1
-rw------- 1 keystore keystore   1576 2021-09-19 16:29 10268_USRPKEY_Key2
-rw------- 1 keystore keystore    727 2021-09-11 15:40 10269_USRCERT_SecureKeyAlias
-rw------- 1 keystore keystore   1683 2021-09-11 15:40 10269_USRPKEY_SecureKeyAlias
-rw------- 1 keystore keystore    727 2021-09-11 16:00 10271_USRCERT_CryptoKeyAlias
-rw------- 1 keystore keystore    727 2021-09-11 15:58 10271_USRCERT_SecureKeyAlias
-rw------- 1 keystore keystore   1683 2021-09-11 16:00 10271_USRPKEY_CryptoKeyAlias
-rw------- 1 keystore keystore   1683 2021-09-11 15:58 10271_USRPKEY_SecureKeyAlias
-rw------- 1 keystore keystore    479 2021-09-20 16:21 10272_USRPKEY_default_key
-rw------- 1 keystore keystore    479 2021-09-20 16:21 10272_USRPKEY_key_not_invalidated
```

Keystore, Keys and KeyBlobs



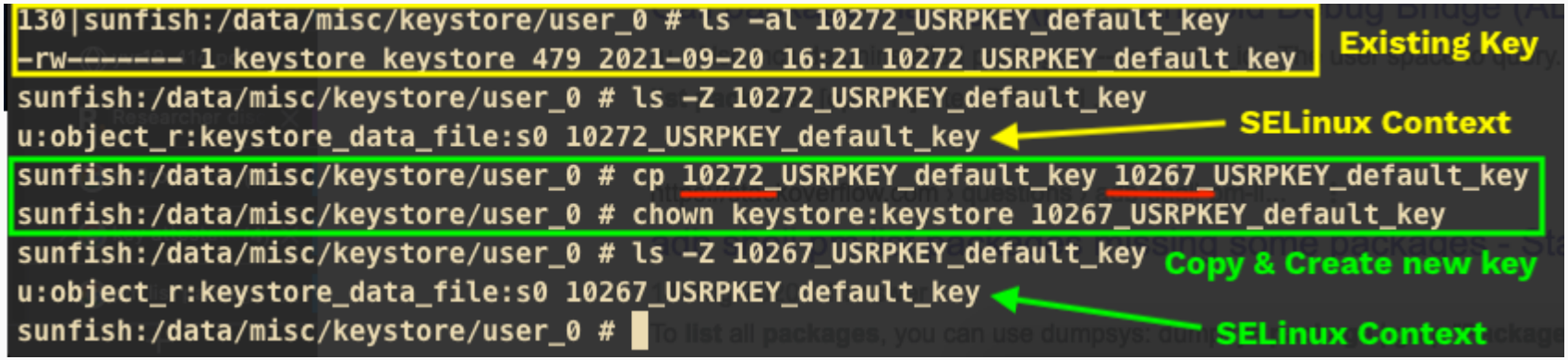
An app requests crypto operations from *keystored* [1]

[1] source: Unearthing the TrustedCore: A Critical Review on Huawei's Trusted Execution Environment

Attack 1: Breaking the App Binding

- Vulnerability - **App UID is not cryptographically bound to the key.**
- A malware with *keystore* user privilege or root privilege.
- Copy the contents of a key and replace the app UID in the filename with the rogue one's.
- Malware can now access & use the key with the same usage properties as set by the original app
  - Can decrypt the encrypted data stored in the original app's sandbox
  - Can sign on behalf of the original app
  - For keys bound to user authentication, can phish the user to authenticate before performing decryption or signing operation

Attack 1


130|sunfish:/data/misc/keystore/user_0 # ls -al 10272_USRPKEY_default_key
-rw------- 1 keystore keystore 479 2021-09-20 16:21 10272_USRPKEY_default_key
Existing Key
sunfish:/data/misc/keystore/user_0 # ls -Z 10272_USRPKEY_default_key
u:object_r:keystore_data_file:s0 10272_USRPKEY_default_key
SELinux Context
sunfish:/data/misc/keystore/user_0 # cp 10272_USRPKEY_default_key 10267_USRPKEY_default_key
sunfish:/data/misc/keystore/user_0 # chown keystore:keystore 10267_USRPKEY_default_key
sunfish:/data/misc/keystore/user_0 # ls -Z 10267_USRPKEY_default_key
Copy & Create new key
u:object_r:keystore_data_file:s0 10267_USRPKEY_default_key
SELinux Context
sunfish:/data/misc/keystore/user_0 #

Attack 1

- DEMO
- Note: Hardware-backed keystore ensures the key cannot be extracted from the device.
  - But it can be used by another malicious application in this case.
- Scenario
  - Private key used for signing a payment request, can create a fake one.
    - A modified original app, installed via malware.
  - App uses symmetric key for local data encryption, can now be decrypted.

Attack 2: The missing attestation

- To bootstrap communication with the server, apps generate key pair in Android Keystore and share the public key with the Server.
- No indication for server that the public key originated from an app it trusts.
- Attack
  - Keystore supports importing key or keypair
  - A malware executing in the context of the application can import a fake keypair when app generates the keypair. Now malware has the control on the keypair.

Attack 2: Demo

- Google Fido Webauthn attack

Comparison And Improvements

Android Keystore vs iOS Keychain

- Broadly both platforms support similar features.

Android	iOS
On application uninstall, keys generated inside the keystore are removed preventing new installed apps with the same UID from accessing those keys.	On application uninstall, keys generated inside the Keychain are not wiped. But the keys can be accessed only by the same application upon re-installation.
Wide range of Cipher algorithms supported except for ECIES.	Supports ECIES and ECDSA algorithms in Secure Enclave.
Supports key access per user authentication or key access within the key validity period.	Supports key access with a valid LA Context till it is invalidated by the user of the key.
Invalidates the key by default upon new biometric enrollment or all biometric data is deleted.	Upon setting kSecAccessControlTouchIDCurrentSet attribute, adding or deleting existing fingerprint invalidates the key.

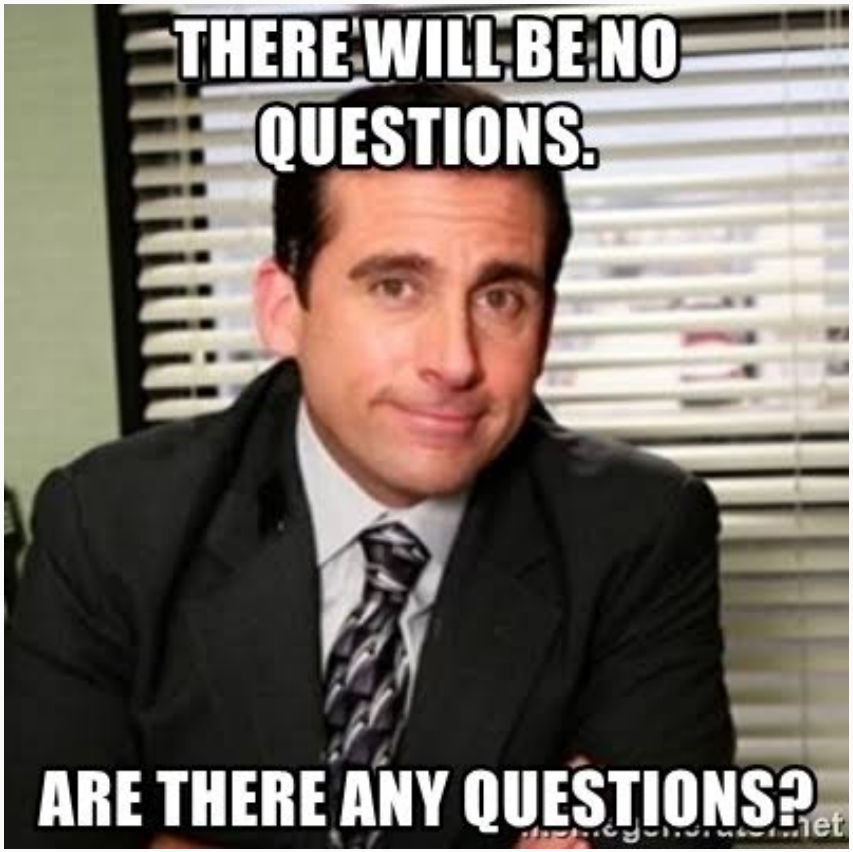
Proposed Improvements

- Cryptographically bind app UID to the key.
  - Attack still possible, but requires more effort.
- Allow application specific password for accessing key/keypairs inside Android keystore
  - Google does not provide it for a reason as developers tend to hardcode the password.
  - There are use cases where app can cryptographically bind the user entered PIN & password to the key
- Keystore access via native code, especially for security sensitive application developed purely in native.
- Support authentication per key access on secure key import.

Takeaways

- Various features of keystore
  - Key attestation
  - Biometric authentication
  - Secure key import
- Attacks
  - App UID binding issue
  - Missing attestation
- Depending on your threat model, keystore usage may still require extra protections/defenses on key usage.
- Comparison with iOS Keychain
- Proposal on improvements

Thank You



References

- Various Android documentation
  - Android keystore system: https://developer.android.com/training/articles/keystore
  - Hardware-backed Keystore: https://source.android.com/security/keystore
- Breaking Into the Keystore: A Practical Forgery Attack Against Android Keystore (Sabt et.al.)
- Analysis of Secure Key Storage Solutions on Android (Cooijmans et.al.)
- Fides: Unleashing the Full Potential of Remote Attestation (Prunster et.al.)
- Key attestation App: https://github.com/vvb2060/KeyAttestation