KOOLSPAN

Smartphone Keystores Compared

ICMC 2016 - Session G11a

May 2016

Bill Supernor, CTO, KoolSpan

Smartphone Keystores Compared

- What is a keystore?
- Points of comparison
- Platforms
  - iOS
  - Android
  - Windows Phone
  - BB10
- Other options

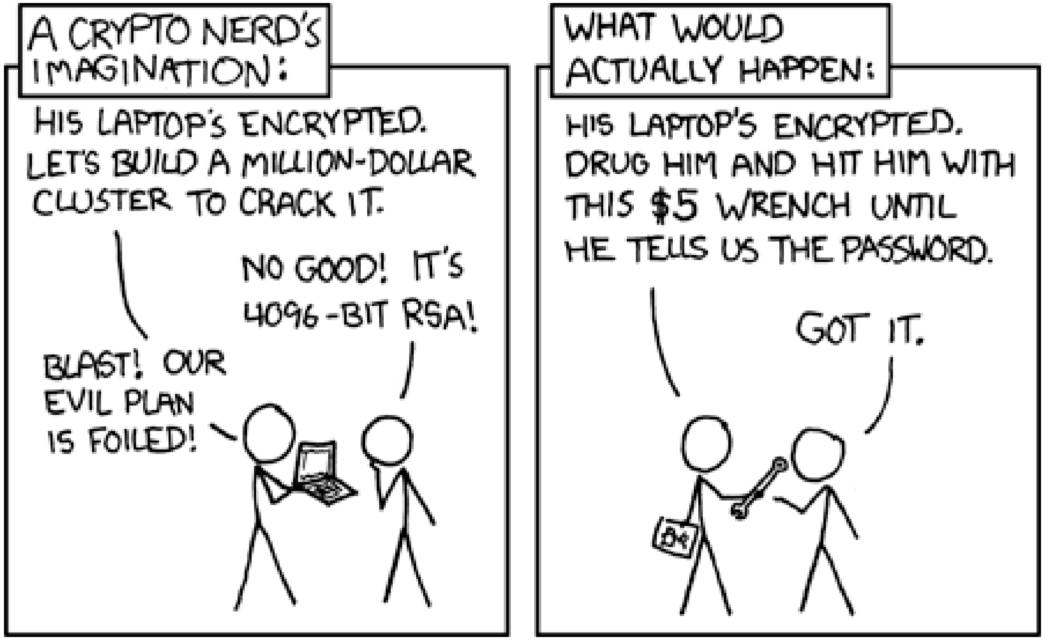
What is a keystore?

- The place in the phone where cryptographic keys and (sometimes) other critical secrets are stored.
- Examples:
  - PKCS#12 files
  - Encrypted databases of key blobs
  - Smartcards/PIV cards
  - Secure microSD devices
  - Other hardware security modules (HSM)
- What's in there?
  - Asymmetric keypairs
  - Symmetric keys
  - Passwords
  - Other secret stuff

From the “Ten Immutable Laws Of Security (Version 2.0)”

(By Scott Culp, Microsoft, 2000)

Law #7: Encrypted data is only as secure as its decryption key.


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.
$4

https://xkcd.com/538/

Law #3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore

What can a keystore do?

- Typical Keystore functions
  - Add/remove key
  - Find key
  - Export key
  - “Use” key in a crypto operation
    - Hopefully by reference - and not by export
- Enforce Access Control Lists (ACLs) on certain functions

How to access - Keystore APIs

- “Standard” interfaces are rare
  - Minimal true cross-platform APIs
  - Standard within a specific platform
  - Cross-platform development always done with an isolation layer
- Java Cryptography Architecture (JCA) and Android APIs
- Apple Keychain
- BlackBerry Certificate Manager API
- MS CAPI
- PKCS11/cryptoki

Where is the keystore?

- A file or database in the file system...hopefully encrypted
- A “protected” part of the device
  - Trusted Execution Environment (TEE)
  - ARM TrustZone
  - Trusted Platform Module (TPM)
- A secure element
  - SIM/UICC card?
  - NFC secure element?
  - Not likely....

How is the keystore protected?

- User, OS, and hardware level defenses
- User
  - “What you know” - User PIN/Password/Pattern
  - “What you are” - Fingerprint
- Hardware/OS defenses
  - OS Secure boot
  - Integrity checks - software and hardware

When are the keys accessible?

- Device unlocked
- Within *x* time of user authentication to device
- Right after boot
- Device locked
  - Some apps require access to keys while device is sleeping/locked

Who can access the keys?

- One user/multiple users
- One app/multiple apps
- One vendor/cross-vendor

OK...so how do they compare?

It’s complicated...

vs.

Features vary by version - *Fragmentation*

- Android (http://developer.android.com/about/dashboards/index.html)
  - Marshmallow - v6: 7.5%
  - Lollipop - v5: 35.6%
  - KitKat - v4.4: 32.5%
  - Jelly Bean - v4.3: 2.9%
  - Everything else: 21.5%
- iOS (https://developer.apple.com/support/app-store/)
  - 9.X: 84%
  - 8.X: 11%
  - Everything else: 5%

Android

- `Keystore` - App-isolated PKI keys
- `KeyChain` - System global visibility
- `KeyChain` uses the `KeyStore` system
- Key file structure highlights user-level KeyChain isolation
  - /data/misc/keystore/user_X, as before (where X is the Android user ID, starting with 0 for the primary user)
  - Encryption of key files depends on Android version and TEE availability
- If keystore not hardware backed, lockscreen password used to derive keys for protecting keystore
- Beyond this...it is version dependent
- OEM information sharing as to implementation details varies widely

Android Keystores - The Older 71%

- Android J
  - AndroidKeyStore Provider - create or store private keys that cannot be used by other applications
  - isBoundKeyType method - allows applications to confirm that system-wide keys are bound to a hardware root of trust for the device (Subsequently deprecated in Android M)
- Android K
  - Some SELinux enforcement, DSA/ECDSA Provider support in AndroidKeyStore
- Android L
  - More SELinux enforcement, TLS with AES-GCM

Marshmallow/v6 - This year’s model...

- Lots more system hardening in core OS
- Major revision to Keystore
  - Supports Symmetric and Asymmetric keys
  - Designed to allow for use of keys without export from Keystore
- New optional key generation parameters
  - Key usage (encr/decr, sign/verify), block mode, padding - stored with key and mandatory for usage in accordance with parms
- Can require authentication on per-key basis and dictate auth validity duration
- Supports complicated crypto operations of potentially arbitrary size with begin/update/finish pattern

Android: Gotchas

- Android Keystore protected by device lock
  - Changing screen lock type (None/PIN/Pattern/PW) wipes keystore in older devices
    - https://code.google.com/p/android/issues/detail?id=61989
  - Android J/v4.3 (2.9%), Android K/v4.4 (32.5%): *Any* lock screen type transition wipes keystore without warning
  - Newer versions of Android warn the user
- A known bug in Android M/v6.0 causes user authentication-related authorizations to be enforced even for public keys in Keystore

iOS Keystore

- Accessed as KeyChain
- Can store passwords, keys, certificates, and blobs
  - With one exception, does not appear to restrict key extraction by apps
- Implemented as a SQLite database stored on the file system
  - Protected with AES-GCM-128 Encryption
  - Not clear if this is on top of the AES-256 file-level Data Protection controlled by Secure Enclave
- Key Item Access Control Lists (ACL)
  - kSecAttrAccessGroup - WHAT app can access key
    - Short version: Keychain items can *only* be shared between apps from the same developer/vendor
  - kSecAttrAccessible - WHEN can the key be accessed
  - kSecAttrAccessControl - What type of authentication is needed

APIs

- Lower-level methods with very granular attribute control
  - `SecItemAdd` to add an item to a keychain
  - `SecItemUpdate` to modify an existing keychain item
  - `SecItemCopyMatching` to find a keychain item and extract information from it
  - `SecItemDelete` to delete an item
- Minimal crypto functions that are actually performed *inside* the keystore
  - Keys have to come up to app space

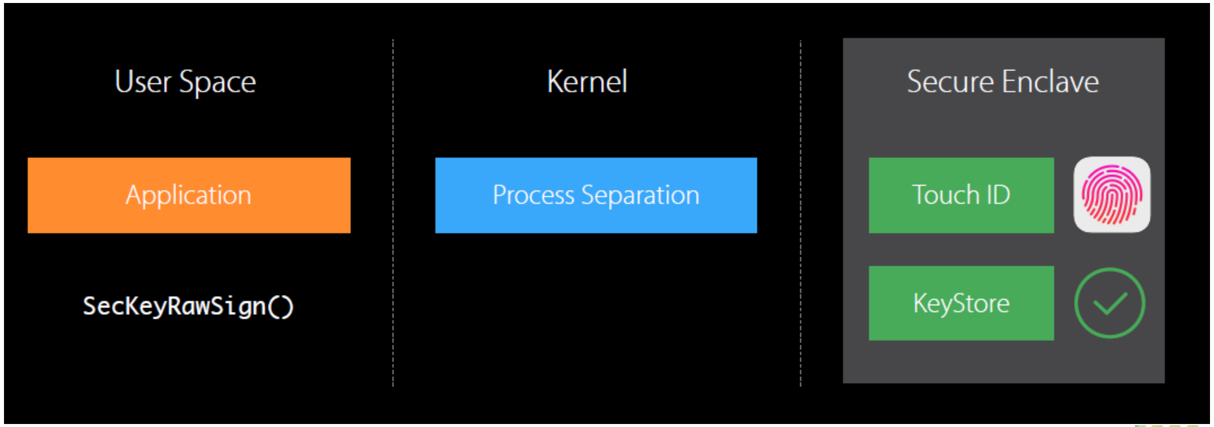
iOS Keychain protection attributes

`kSecAttrAccessible` ACL's

Data Protection	Availability
`kSecAttrAccessibleAfterFirstUnlock` ★	Key inaccessible after boot until user enters passcode for 1st time (recommended for background services)
`kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` ★	Same as above...but cannot be backed up and then restored to a different device
`kSecAttrAccessibleAlways`	Key accessible anytime after boot (deprecated in iOS 9)
`kSecAttrAccessibleAlwaysThisDeviceOnly`	Same as above...but...
`kSecAttrAccessibleWhenUnlocked` ★	DEFAULT mode. Key accessible when device unlocked
`kSecAttrAccessibleWhenUnlockedThisDeviceOnly`	Same as above...but...
`kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`	Added in iOS 8. Same as above, but password MUST exist. ★

iOS Secure Enclave

- iPhone 5s and later
- Processor for TouchID and KeyStore
  - Basically: ARM TrustZone
  - Stores its own data in device storage but uniquely keyed and unknown to ANYONE
- Can generate/store/use unexportable EC P256 key
  - Enables protected calls to `SecKeyRawSign()` and `SecKeyRawVerify()`
  - Preservation of the associated public key left as an exercise for the student...



iOS TouchId

- Biometric user authentication
- Hardware sensor and Secure Enclave get pre-shared secret at Mfg time
- Provides further granularity to key access and bind a credential more closely to Touch ID
- Used with attribute `kSecAttrAccessControl`

Attribute	Control
`UserPresence`	Require TouchID and fallback to passcode
`TouchIDAny`	TouchId with no fallback
`TouchIDCurrentSet`	Only allows access if enrolled TouchID has not changed since item stored Someone with device passcode *cannot* login, add finger to TouchID, and then access credential
`DevicePasscode`	Passcode only
`ApplicationPassword`	Password from App required to decrypt credential Password entered by user or perhaps from a live server
`PrivateKeyUsage`	Leverage asymmetric private key that never leaves the KeyStore EC P256, supporting sign and verify

iOS other tidbits/gotchas

- Watch out for iCloud Keychain
  - Passwords/keys can be shared across devices
  - Set attribute kSecAttrSynchronizable to false to prevent sync
- Keys cannot be shared between apps from different vendors
  - Complications for provisioning derived credentials
  - DISA “Purebred” solution?
- iPhone “memory pressure” issue - key access denied (https://forums.developer.apple.com/message/116056)
- Items written to Keychain are not removed when app uninstalled

Windows Phone Keystore

- Two more or less distinct keystores
- Credential Locker
  - Apps can only access their own credentials
  - Credentials “roam” between a user’s devices along with the user Microsoft account
- Virtual Smart Card
  - Keys are bound to the hardware and can only be accessed when user PIN is provided
  - Potentially more “traditional” Derived Credential approach
  - Built on top of TPM
- All Windows Phone 8.1 devices include a TPM (Trusted Platform Module)
  - TPM used to protect cryptographic calculations, virtual smart cards, and certificates

BlackBerry Keystore

- More specifically...BlackBerry 10
- Keys managed by BlackBerry Certificate Manager API
- This is pretty easy...
  - Unless you are talking about the *native* Email, VPN, or Browser apps...
  - ...and about importing PKCS#12 files...
- There is no native keystore capability for 3rd party vendors
  - “Should be in a forthcoming release” ☹ ☹ ☹ ☹
  - Right now only supports secure password storage
- The good news...Android-based BB Priv is pretty solid

Keystores and FIPS

- Which keystores use or provide FIPS 140-2 validated crypto?
- Not 100% clear...but...
  - Windows Phone - Definitely
  - Apple - Very Probably
  - Android (at least Samsung) - Maybe
  - BlackBerry 10 - Definitely not
- Caveat #1: All are FIPS 140-2 Level 1
- Caveat #2: Lots of OpenSSL deployed with mobile OS’s...some *could* be FIPS.
- On my wish list: every keystore and crypto implementation should provide a version API that includes “I am in FIPS mode.”
  - Frequently difficult to correlate evaluated module name with where it is used in an OS, especially when it is KNOWN that an OS has multiple crypto modules.

Other options

- What if FIPS 140-2 Level 1 is not good enough?
- Smart cards?
  - Tethered or Bluetooth sleds are cumbersome
  - Device-tailored cases/sleeves cannot keep up with device shape
  - NFC-based smartcards would be a great option
- Secure microSD devices
  - PKI Smart Card in a microSD form factor
  - Provide PKCS#11 or full ISO 7816 APDU interfaces
  - Provide standalone hardware security modules
  - Fairly well-supported across Android, iOS, BB10
  - iOS requires adapters...which brings us back to smart card challenges
  - Overall: a potential solution when higher grade crypto is essential

Parting thoughts...

- Market fragmentation makes availability of key features unpredictable (pun intended)
- Different platforms have different strengths
- Disparate API's/features makes writing common key management a challenge
- Mobile keystores continue to evolve in a generally positive direction
  - Improving in strength and features

Awesome references

- https://nelenkov.blogspot.com/2015/06/keystore-redesign-in-android-m.html
- http://www.samsung.com/hk_en/business-images/insights/2015/Android_security_maximized_by_Samsung_KNOX_0315_online-0.pdf
- https://www.apple.com/business/docs/iOS_Security_Guide.pdf
- https://github.com/Purebred
- http://video.ch9.ms/sessions/teched/na/2014/WIN-B220.pptx (TechEd - Windows Phone 8.1 Security for Developers)
- https://www.cs.ru.nl/E.Poll/papers/AndroidSecureStorage.pdf (Analysis of Secure Key Storage Solutions on Android)
- https://developer.android.com/training/articles/keystore.html

KOOLSPAN

Thank you!

Merci!

Contact: bsupernor@koolspan.com