Smartphone Keystores

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

Law #7: Encrypted data is only as secure as its decryption key.
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)
  • Dedicated processor
• 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](http://developer.android.com/about/dashboards/index.html))

<table>
<thead>
<tr>
<th>Version</th>
<th>Percentage</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nougat v7+7.1</td>
<td>7.1%</td>
<td>(+7.1%)</td>
</tr>
<tr>
<td>Marshmallow v6</td>
<td>31.2%</td>
<td>(+23.7%)</td>
</tr>
<tr>
<td>Lollipop v5+5.1</td>
<td>32.0%</td>
<td>(-3.6%)</td>
</tr>
<tr>
<td>KitKat v4.4</td>
<td>18.8%</td>
<td>(-13.7%)</td>
</tr>
<tr>
<td>Jelly Bean v4.3</td>
<td>1.3%</td>
<td>(-1.6%)</td>
</tr>
<tr>
<td>Everything else...</td>
<td>9.6%</td>
<td>(-11.9%)</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Version</th>
<th>Percentage</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.x</td>
<td>79%</td>
<td>(+79%)</td>
</tr>
<tr>
<td>9.x</td>
<td>16%</td>
<td>(-68%)</td>
</tr>
<tr>
<td>Everything else...</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>
Android Keystore

- **Keystore** - App-isolated PKI keypairs
- **KeyChain** - Special instance of Keystore with System global visibility
- **KeyMaster** - Hardware Abstraction Layer (HAL) for encryption of keys
- Keys stored in flat files, highlighting user-and-app-level KeyChain isolation
  - `/data/misc/keystore/user_X/AppUID_keyname`, 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 with PBKDF
- Beyond this...it is version dependent
- Most OEMs use ARM TrustZone-based keystores - many on QSEE or Trustonic TEE
• Android J (v4.1, 4.2, 4.3)
  • AndroidKeyStore Provider - create/import/store/use(sign+verify) private RSA keys, not usable by other apps
  • 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)
  • As of 4.2: default SecureRandom provider is OpenSSL.

• Android K (v4.4)
  • AndroidKeyStore adds support for EC keys + DSA/ECDSA
  • SecretKeyFactory with PBKDF2WithHmacSHA1 uses all available bits of Unicode passphrase per PKCS #5.

• Android L (v5.x)
  • TLS with AES-GCM
Marshmallow/v6  (31.2%)

• Major revisions to Keystore + Keymaster
  • Support for symmetric keys + primitives
  • Access control system for specific users, apps, time ranges
  • Key usage restrictions - 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
Nougat/v7 - This year’s model... (7.1%)

- Relevant core OS hardening:
  - Verified Boot now strictly enforced to prevent compromised devices from booting - and blocks access to the keystore.
  - Hardware-backed keystore mandatory (TEE or better)
  - User and MDM-installed root CA’s no longer globally trusted by default...APIs added to enable trust.
  - Cross-OEM-standardized trusted CAs

- RNG changes:
  - SHA1PRNG algorithm and “Crypto” provider deprecated
    - SecureRandom.getInstance("SHA1PRNG", "Crypto") Will only work for M and below
  - If SHA1PRNG is requested without explicit Provider, OS will return an instance of OpenSSLRandom.
Android: Gotchas

• Android Keystore protected by device lock
  • Changing screen lock type (None/PIN/Pattern/PW) wipes keystore in older devices
  • The bug: https://issuetracker.google.com/issues/36983155
    • (Or for detailed analysis: https://doridori.github.io/android-security-the-forgetful-keystore/ )
  • Android J, K, and some M: *Any* lock screen type transition wipes keystore without warning
  • Newer versions of Android survive *most* transitions or warn the user if the particular transition will wipe the keys.

• Restricted access to public keys in Android M/v6.0
  • ACL rules set for private key also get applied to public keys (Workaround: extract/store public key material outside keystore)
iOS Keychain

- Can store passwords, keys, certificates, and blobs
  - With one exception, does not appear to restrict key extraction by apps
- Implemented as a single SQLite database stored on the file system, owned by `securityd`
- 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
- ACL decisions are made in the Secure Enclave Processor
- Keychains can be collected and managed in groupings called “Keybags”
iOS Secure Enclave Processor (SEP)

- iPhone 5s and later
  - A distinct processor + kernel inside the SoC for TouchID and KeyStore
  - *Distinct* from the main CPU’s ARM TrustZone (which appears to be dedicated to Kernel Patch Protection)
  - Stores its own data in device storage but uniquely keyed and unknown to ANYONE
  - May be used to protect KeyChain items via TouchID or device password

- Can generate/store/use unexportable EC P256 keys
  - Enables protected calls to `SecKeyRawSign()` and `SecKeyRawVerify()`
  - Preservation of the associated public key left as an exercise for the student...

*From Keychain and Authentication with Touch ID - WWDC14*
# iOS Keychain protection attributes

- **kSecAttrAccessible** ACL’s control when a key can be accessed

<table>
<thead>
<tr>
<th>Data Protection</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>kSecAttrAccessibleAfterFirstUnlock</td>
<td>After user enters passcode for 1st time after reboot (recommended for background services)</td>
</tr>
<tr>
<td>kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly</td>
<td>Same as above...but cannot be backed up to iCloud and then restored to a different device</td>
</tr>
<tr>
<td>kSecAttrAccessibleAlways</td>
<td>Key accessible anytime after boot (deprecated in iOS 9)</td>
</tr>
<tr>
<td>kSecAttrAccessibleAlwaysThisDeviceOnly</td>
<td>Same as above...but...</td>
</tr>
<tr>
<td>kSecAttrAccessibleWhenUnlocked</td>
<td>DEFAULT mode. Key accessible when device unlocked</td>
</tr>
<tr>
<td>kSecAttrAccessibleWhenUnlockedThisDeviceOnly</td>
<td>Same as above...but...</td>
</tr>
<tr>
<td>kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly</td>
<td>Added in iOS 8. Key accessible when device unlocked, but password MUST exist. NEVER backed-up.</td>
</tr>
</tbody>
</table>
iOS Roots of Trust and RNG

- **UID** - 256-bit Unique ID/key - generated in SEP at Mfg
  - Used in file system encryption key hierarchy
- **GID** - 256-bit Group ID/key - inserted in SEP at Mfg
  - Common across all devices in a processor family
  - Firmware encryption
- iOS devices use a feature called Effaceable Storage to securely erase critical keys from NAND
  - Bypasses NAND wear-leveling to directly address and erase a small number of blocks at a very low level
- SEP includes a “true hardware random number generator based on multiple ring oscillators post processed with CTR_DRBG”
- All other cryptographic keys are created in the OS using an algorithm based on CTR_DRBG.
• 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 10 `CryptoTokenKit` API adds native support for Smart Cards and USB crypto tokens
  • iOS 10 also added APIs and algorithms for asymmetric cryptographic operations which are now unified across iOS and macOS
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`

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

• Watch out for iCloud Keychain
  • Some passwords/keys can be shared across devices
  • Set attribute kSecAttrSynchronizable to false to prevent sync or use ...ThisDeviceOnly ACL

• Keys cannot be shared between apps from different vendors
  • Complications for provisioning derived credentials for use by apps from multiple vendors

• iPhone “memory pressure” issue - key access denied (https://forums.developer.apple.com/message/185130)

• Items written to Keychain are not removed when app uninstalled
Windows Phone/Mobile 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
- TPM (Trusted Platform Module) mandatory in Windows Phone 8.1 and Windows 10 Mobile
  - Protect cryptographic calculations, virtual smart cards, and certificates
- Native support for biometrics
BlackBerry 10 Keystore

• Keys managed by BlackBerry Certificate Manager API
  • Keystore is implemented with ARM TrustZone
  • Supports PKI (with caveats) and passwords
  • Permits binding of items to User, App, or Enterprise (aka, BES)
  • Allows blocking export/backup of private keys
  • Appears to support user password prompting to unlock keystore

• **BUT** ...the PKI keystore is only available to native Email, VPN, Browser apps
  • There is no native PKI keystore capability for 3rd party vendors
  • Right now only supports secure password storage
Keystores and FIPS

• Which keystores use or provide FIPS 140-2 validated crypto?
  • Windows Phone - Definitely
  • Apple - Definitely
  • Android - It depends… (Samsung flagships - probably)
  • BlackBerry 10 - Definitely

• Caveat #1: All are FIPS 140-2 Level 1

• Caveat #2: Lots of OpenSSL deployed with mobile OS’s…some probably FIPS. (Samsung using BoringSSL fork)

• Caveat #3: Exceedingly difficult to determine if crypto used by OS is running in FIPS Mode, as APIs are buried.
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 changes
  • NFC-based smartcards would be a great option
• Secure microSD devices
  • PKI Smart Card in a microSD form factor (Such as GoTrust)
  • FIPS 140-2 Level 3
  • Provide PKCS#11 or full ISO 7816 APDU interfaces
  • Supported on iOS and Android
  • 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
• 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
• https://developer.android.com/training/articles/keystore.html
Thank you!

Contact: bsupernor@koolspan.com