Evolution of iOS Data Protection and iPhone Forensics: from iPhone OS to iOS 5

Andrey Belenko & Dmitry Sklyarov
Elcomsoft Co. Ltd.
Agenda

• Basics
• iOS Security before iOS 4
• iOS 4 Data Protection
• iOS 5 Data Protection Changes
• Summary
GOALS:

1. Assuming physical access to the device extract as much information as practical

2. Leave as little traces/artifacts as practical
iOS: Why Even Bother?

- Almost 5 years on the market
- 250+ million iOS devices sold worldwide
- 6 iPhones, 4 iPods, 2 iPads
- “Smart devices” – they do carry a lot of sensitive data
- Corporate deployments are increasing

There was, is, and will be a real need in iPhone Forensics
iPhone Forensics 101

• Passcode
  – Prevents unauthorized access to the device
  – Bypassing passcode is usually enough

• Keychain
  – System-wide storage for sensitive data
  – Encrypted

• Storage encryption
• Logical: iPhone Backup
  – Ask device to produce a backup
    – Device must be unlocked
    – Device may produce encrypted backup
    – Limited amount of information

• Physical: filesystem acquisition
  – Boot-time exploit to run unsigned code
    – Device lock state isn’t relevant
    – Can get all information from the device
The Inception

Runs iPhone OS (up to 3.1.3)
• Based on Mac OS X

Has a crypto co-processor

06/29/2007
iPhone
Two embedded AES keys:

- **GID** – shared by all devices of same family
- **UID** – unique for each and every device

*No known ways to extract GID/UID keys*
Device Keys

• To avoid unnecessary exposure, usage of UID/GID keys is limited

• Device keys are computed from hardware keys during boot:
  
  - 0x835 = AES_Enc (UID, 01010101010101010101010101010101);
  - 0x836 = AES_Enc (UID, 00E5A0E6526FAE66C5C1C6D4F16D6180);
  - 0x837 = AES_Enc (GID, 345A2D6C5050D058780DA431F0710E15);
  - 0x838 = AES_Enc (UID, 8C8318A27D7F030717D2B8FC5514F8E1);
iPhone OS Security

Relies on chain of trust:
• BootROM loads trusted iBoot
• iBoot loads trusted kernel
• Kernel runs trusted apps

Apps must be signed
• Developers can sign and run their apps on their devices ($99/yr)

Applications are sandboxed
• Jailbreak – circumventing iOS security in order to run custom code
• Boot-level or application-level
• Tethered or untethered
Breaking Free

• App-level JB gets kernel code execution by exploiting apps
  – e.g. JailbreakMe
  – Can be fixed by new firmware

• Boot-level JB breaks loads custom kernel by breaking chain of trust
  – e.g. limera1n
  – Can’t be fixed if exploits vulnerability in BootROM
Jailbreak + Forensics = ?

• Tethered JB
  – Host connection is required to boot into JB state
  – Exploit(s) are sent by the host
  – May leave minimal traces on the device

• Untethered JB
  – Device is modified so that it can boot in jailbroken state by itself
  – Leaves permanent traces
iPhone OS Passcode

- Lockscreen (i.e. UI) is the only protection
- Passcode is stored in the keychain
  - Passcode itself, not its hash
- Can be recovered or removed instantly
  - Remove record from the keychain
  - And/or remove setting telling UI to ask for the passcode
iPhone OS Keychain

- SQLite3 DB, only passwords are encrypted
- All items are encrypted with the device key (0x835) and random IV
- Key can be extracted (computed) for offline use
- All past and future keychain items from the device can be decrypted using that key
iPhone OS Storage Encryption

• No encryption.
iPhone 3G

Hardware is very similar to original iPhone

No real security improvements over previous model

06/29/2007 iPhone
07/11/2008 iPhone 3G
iPhone 3GS

New application processor

Hardware storage encryption
iPhone 3GS Forensics

• Passcode: same as before
• Keychain: same as before
• Storage encryption:
  – Only user partition is encrypted
  – Single key for all data (FDE)
  – Designed for fast wipe, not confidentiality
  – Transparent for applications
  – Does not affect physical acquisition

This is true only for iPhone 3GS running iPhone OS 3.x
No notable enhancements in security hardware over iPhone 3GS

Shipped with iOS 4 with major security improvements
• Basics
• iOS Security before iOS 4
• iOS 4 Data Protection
• iOS 5 Data Protection Changes
• Summary

Dmitry Sklyarov
iOS 4 Data Protection

• More robust passcode protection
• Better storage encryption
  – Metadata is encrypted transparently (same as before)
  – Per-file encryption keys
• Better Keychain encryption
• New backup format
  – Slower password recovery
  – Keychain items can migrate to another device
Protection Classes

- Content grouped by accessibility requirements:
  - Available only when device is unlocked
  - Available after first device unlock (and until power off)
  - Always available

- Each protection class has a master key

- Master keys are protected by device key and passcode

- Protected master keys form system keybag
  - New keys created during device restore
Effaceable Storage

• Special region of flash memory to store small data items with ability to quickly erase them
• Items within effaceable storage are called lockers
• As of iOS 4: 960 bytes capacity, 3 lockers:
  – ‘BAG1’ – System Keybag payload key and IV
  – ‘Dkey’ – NSProtectionNone class master key
  – ‘EMF!’ – Filesystem encryption key
System Keybag

• /private/var/keybags/systembag.kb
• Three layers of encryption:
  – System keybag file is encrypted by Data Protection
  – Keybag payload is encrypted before writing to disk
  – Master keys are encrypted with device key and/or passcode key
Escrow Keybag

• “Usability feature” to allow iTunes to unlock the device
• Contains same master keys as system keybag
• Stored on the iTunes side
• Protected by 256 bit random “passcode” stored on the device
• With iOS 4, escrow keybag gives same powers as knowing the passcode
Backup Keybag

• Included in the iOS backups
• Holds keys to decrypt files and keychain items included with the backup
• New keys are generated for each backup
Unlocking Keybag

Keybag (locked)

Protected Key
WRAP = 1

Protected Key
WRAP = 2

Protected Key
WRAP = 3

Protected Key
WRAP = 1

Protected Key
WRAP = 3

...

Keybag (unlocked)

Passcode Key
if (WRAP & 0x2)

Device Key
if (WRAP & 0x1)

UNWRAP

UNWRAP

UNWRAP

DECRIPT

DECRIPT

DECRIPT

...

Key

Key

Key

Key

Key

...

if (WRAP & 0x2)

if (WRAP & 0x1)
iOS 4 Passcode

• Passcode is used to compute passcode key
  – Computation tied to hardware key
  – Same passcode will yield different passcode keys on different devices!

• Passcode key is required to unlock most keys from the system keybag
  – Most files are protected with NSProtectionNone and don’t require a passcode
  – Most keychain items are protected with ...WhenUnlocked or ...AfterFirstUnlock and require a passcode
• Passcode-to-Key transformation is slow
• Offline brute-force currently is not possible
  – Requires extracting hardware key
• On-device brute-force is slow
  – 2 p/s on iPhone 3G, 7 p/s on iPad
• System keybag contains hint on password complexity
iOS 4 Passcode

- 0 – digits only, length = 4 (simple passcode)
iOS 4 Passcode

- 0 – digits only, length = 4 (simple passcode)
- 1 – digits only, length != 4
iOS 4 Passcode

- 0 – digits only, length = 4 (simple passcode)
- 1 – digits only, length ≠ 4
- 2 – contains non-digits, any length
iOS 4 Passcode

- 0 – digits only, length = 4 (simple passcode)
- 1 – digits only, length != 4
- 2 – contains non-digits, any length

Can at least identify weak passcodes
iOS 4 Keychain

• SQLite3 DB, only passwords are encrypted

• Available protection classes:
  – kSecAttrAccessibleWhenUnlocked (+ ...ThisDeviceOnly)
  – kSecAttrAccessibleAfterFirstUnlock (+ ...ThisDeviceOnly)
  – kSecAttrAccessibleAlways (+ ...ThisDeviceOnly)

• Random key for each item, AES-CBC

• Item key is protected with corresponding protection class master key
iOS 4 Storage

• Only User partition is encrypted

• Available protection classes:
  – NSProtectionNone
  – NSProtectionComplete

• When no protection class set, EMF key is used
  – Filesystem metadata and unprotected files
  – Transparent encryption and decryption (same as pre-iOS 4)

• When protection class is set, per-file random key is used
  – File key protected with master key is stored in extended attribute com.apple.system.cprotect
No known security enhancements in hardware over iPhone 4

Shipped with iOS 5 with some security improvements
• Basics
• iOS Security before iOS 4
• iOS 4 Data Protection
• iOS 5 Data Protection Changes
• Summary
iOS 5 Passcode

• Very similar to iOS 4
• Passcode key computation utilizes new hardware key UID+
  – UID is used instead of UID+ on devices before iPad 2 and iPhone 4S
  – Not clear for iPad 2 and iPhone 4S
### iOS 5 Keychain

- All attributes are now encrypted (not only password)
- AES-GCM is used instead of AES-CBC
  - Enables integrity verification

<table>
<thead>
<tr>
<th></th>
<th>Class</th>
<th>Wrapped Key Length</th>
<th>Wrapped Key</th>
<th>Encrypted Data (+Integrity Tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>
iOS 5 Storage

• New partition scheme
  – “LwVM” – Lightweight Volume Manager
• Any partition can be encrypted
• New protection classes
  – NSFileProtectionCompleteUntilFirstUserAuthentication
  – NSFileProtectionCompleteUnlessOpen
• IV for file encryption is computed differently
Creating the File

NSFileProtectionCompleteUnlessOpen

Generate random file key (AES)

Generate file public/private keys (ECC)

Master key from the system keybag (ECC)

\[ K_F \]

\[ \text{Encrypt} \]

\[ \text{com.apple. system. cprotect} \]

\[ \text{Shared Secret} \]

\[ Pub_F \]

\[ Priv_F \]

\[ Pub_{KB} \]

\[ Priv_{KB} \]

- Generate random file key (AES)
- Generate file public/private keys (ECC)
- Master key from the system keybag (ECC)
Reading the File
NSFileProtectionCompleteUnlessOpen

File key (AES)

File public/private keys (ECC)

Master key from the system keybag (ECC)

Requires a passcode (if any)

com.apple.system.cprotect

Decryption

Shared Secret

$K_F$

$Pub_F$

$Priv_F$

$Pub_{KB}$

$Priv_{KB}$
Reading the File

NSFileProtectionCompleteUnlessOpen

File key (AES)

File public/private keys (ECC)

Master key from the system keybag (ECC)

Requires a passcode (if any)

Looks pretty much like BlackBerry way to receive emails while locked :-)

com.apple.system.cprotect
• Acquiring disk image is not enough for iOS 4+
  – Content protection keys must also be extracted from the device during acquisition
  – Effaceable Storage contents are also needed to decrypt dd images.

• Passcode or escrow keybag is needed for a complete set of master keys

• In real world it might be a good idea to extract source data and compute protection keys offline
Must be done on the device
Required to decrypt files/keychain
Sufficient for offline key reconstruction

**Effaceable Storage**

- ‘EMF!’ / ‘LwVM’
- ‘Dkey’
- ‘BAG1’

**System Keybag (locked)**

| Class Key #1
| Class Key #2
| Class Key #3
| Class Key #4
| Class Key #5
...
| Class Key #11

Unlock

**System Keybag (unlocked)**
<table>
<thead>
<tr>
<th>Physical acquisition</th>
<th>Phone 1</th>
<th>Phone 3G</th>
<th>Phone 3GS</th>
<th>Phone 4</th>
<th>Phone 4S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passcode recovery</td>
<td>instant</td>
<td>+ instant</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Keychain decryption</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Disk decryption</td>
<td>not encrypted</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Conclusions

- iPhone physical analysis is possible
- Physical acquisition requires boot-time exploit
- Passcode is *usually* not a problem
  - Due to technology before iOS 4
  - Due to human factor with iOS 4/5
- Both proprietary and open-source tools for iOS 4/5 acquisition are available
THANK YOU

QUESTIONS?
Please do not forget to complete your feedback survey forms!