iPhone data protection in depth

Jean-Baptiste Bédruine
Jean Sigwald
Sogeti / ESEC
jean-baptiste.bedrune(at)sogeti.com
jean.sigwald(at)sogeti.com
Introduction

Motivation

- Mobile privacy is a growing concern
- iPhone under scrutiny
  - iPhoneTracker (O'Reilly)
  - “Lost iPhone? Lost Passwords!” (Fraunhofer)

Agenda

- iOS 4 data protection
- Storage encryption details
- iTunes backups
iPhone forensics

**Trusted boot vulnerabilities**

- Chain of trust starting from BootROM
- BootROM runs USB DFU mode to allow bootstrapping of restore ramdisk
- Unsigned code execution exploits through DFU mode
  - Pwnage/steaks4uce/limera1n (dev team/pod2g/geohot)
  - All devices except iPad 2

**Custom ramdisk techniques**

- Zdziarski method, msft_guy ssh ramdisk
- Modify ramdisk image from regular firmware, add sshd and command line tools
- Boot (unsigned) ramdisk and kernel using DFU mode exploits
- Dump system/data partition over usb (usbmux)
Embedded AES keys

- **UID key**: unique for each device
- **GID key**: shared by all devices of the same model
  - Used to decrypt IMG3 firmware images (bootloaders, kernel)
  - Disabled once kernel boots
- **IOAESAccelerator kernel extension**
  - Requires kernel patch to use UID key from userland

UID key

- Encrypts static nonces at boot to generate unique device keys
  - key0x835 = AES(UID, "01010101010101010101010101010101")
  - key0x89B = AES(UID, "183e99676bb03c546fa468f51c0cbd49")
- Also used for passcode derivation in iOS 4
iOS 3.x data protection

Hardware Flash memory encryption

- Introduced with iPhone 3GS
- Allows fast remote wipe
- Data still accessible transparently from custom ramdisk

Keychain

- SQLite database for passwords, certificates and private keys
- Each table has an encrypted data column
- All items encrypted with key 0x835
- Format: IV + AES128(key835, data + SHA1(data), iv)
Data protection

- Set of features to protect user data
- Phone passcode used to protect master encryption keys
- Challenges for iOS 4 forensics:
  - Keychain encryption has changed
  - Some protected files cannot be recovered directly from custom ramdisk
  - Raw data partition image cannot be read with standard tools
  - New encrypted iTunes backup format

Our work

- Keychain tools
- Passcode bruteforce
- Data partition encryption scheme
- iTunes backup tools
Plan

1 Introduction

2 Data protection
   Overview
   System & Escrow keybags
   Keychain
   Passcode derivation
   Bruteforce attack

3 Storage encryption

4 iTunes Backups

5 Conclusion
Data protection

Objectives

- Protect data at rest (phone locked or powered off)
  - Limit impact from custom ramdisk attacks
- Encrypted data protected by user’s passcode
  - Limit bruteforce attacks speed with custom passcode derivation function

Design

- Data availability
  - When unlocked
  - After first unlock
  - Always
- Protection Classes for files and keychain items
- Master keys for protection classes stored encrypted in a keybag
  - 3 keybag types: System, Escrow, Backup
Data protection

### Protection classes

<table>
<thead>
<tr>
<th>Availability</th>
<th>Filesystem</th>
<th>Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>When unlocked</td>
<td>NSProtectionComplete</td>
<td>WhenUnlocked</td>
</tr>
<tr>
<td>After first unlock</td>
<td></td>
<td>AfterFirstUnlock</td>
</tr>
<tr>
<td>Always</td>
<td>NSProtectionNone</td>
<td>Always</td>
</tr>
</tbody>
</table>

### Implementation

- keybagd daemon
- AppleKeyStore kernel extension
  - MobileKeyBag private framework (IOKit user client)
- AppleKeyStore clients:
  - Keychain
  - HFS content protection (filesystem)
Data protection components & interactions

- AppleEffaceableStorage
- AppleKeyStore
- KeyBagCreateWithData
- KeyBagSetSystem
- KeyWrap/KeyUnwrap
- HFS content protection
- getLocker BAG1
- LockDevice/UnlockDevice
- Lock state notifications
- systembag.kb
- keybagd
- securityd
- Applications
- F_SETPROTECTIONCLASS
- Keychain API
- kernel
- userland
- file keys
Keybagd

Description

- System daemon, loads system keybag into AppleKeyStore kernel service at boot
- Handles system keybag persistence and passcode changes

System keybag

- Stored in `/private/var/keybags/systembag.kb`
- Binary plist with encrypted payload
- Encryption key pulled from AppleEffaceableStorage kernel service
  - Stored in “BAG1” effaceable locker
- Tag-Length-Value payload
Keybag binary format

<table>
<thead>
<tr>
<th>Example keybag hexdump</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000: 4441 5441 0000 0444 5645 5253 0000 0004</td>
</tr>
<tr>
<td>0000010: 0000 0002 5459 5045 0000 0004 0000 0000</td>
</tr>
<tr>
<td>0000020: 5555 4944 0000 0010 ceea c20d cf52 40e0</td>
</tr>
<tr>
<td>0000030: ac0e dd52 915d 38bc 484d 434b 0000 0028</td>
</tr>
<tr>
<td>0000040: 6785 4e94 bc50 f2e4 541b c51d 8f46 ad59</td>
</tr>
<tr>
<td>0000050: 3af3 cdc6 201a 2e53 6424 b728 3775 788f</td>
</tr>
<tr>
<td>0000060: cd2e 28f8 b692 2bac 5752 4150 0000 0004</td>
</tr>
<tr>
<td>0000070: 0000 0001 5341 4c54 0000 0014 8bda 11d7</td>
</tr>
<tr>
<td>0000080: 43bb 669c e451 646c 2ea9 ac0b 6658 ff9d</td>
</tr>
<tr>
<td>0000090: 4954 4552 0000 0004 0000 c350 5555 4944</td>
</tr>
<tr>
<td>00000a0: 0000 0010 02ed b2ea c187 49b2 b9f1 7925</td>
</tr>
<tr>
<td>00000b0: ddab daae 434c 4153 0000 0004 0000 000b</td>
</tr>
<tr>
<td>00000c0: 5752 4150 0000 0004 0000 0001 5750 4b59</td>
</tr>
<tr>
<td>00000d0: 0000 0020 8f81 980c a483 2ae4 e978 4cc8</td>
</tr>
<tr>
<td>00000e0: f715 f4e3 44ac 71cc b568 22e6 e119 6983</td>
</tr>
<tr>
<td>00000f0: b156 e25e 5555 4944 0000 0010 d8e0 f7a2</td>
</tr>
</tbody>
</table>
Keybag binary format

**Header**

- **VERS**: 1 or 2
  - Version 2 was introduced in iOS 4.3
  - Minor changes in passcode derivation function
- **TYPE**: Keybag type
  - 0: System
  - 1: Backup
  - 2: Escrow
- **UUID**, **ITER**, **SALT**, **WRAP**
- **HMCK**: encrypted HMAC key for integrity check
- **SIGN** = HMAC_SHA1(DATA, AES_UNWRAP(key835, HMCK))
  - HMAC parameters inverted, DATA is the HMAC key (?!)

**Overview**
- System & Escrow keybags
- Keychain
- Passcode derivation
- Bruteforce attack
Keybag binary format

Wrapped class keys

- UUID: Key uuid
- CLAS: Class number
- WRAP: Wrap flags
  - 1: AES encrypted with key 0x835
  - 2: AES wrapped with passcode key (RFC 3394)
- WPKY: Wrapped key
### Class keys identifiers

<table>
<thead>
<tr>
<th>Id</th>
<th>Class name</th>
<th>Wrap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NSProtectionComplete</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>(NSFileProtectionWriteOnly)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>(NSFileProtectionCompleteUntilUserAuthentication)</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>NSProtectionNone (stored in effaceable area)</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>unused ? (NSFileProtectionRecovery ?)</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>kSecAttrAccessibleWhenUnlocked</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>kSecAttrAccessibleAfterFirstUnlock</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>kSecAttrAccessibleAlways</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>kSecAttrAccessibleWhenUnlockedThisDeviceOnly</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>kSecAttrAccessibleAlwaysThisDeviceOnly</td>
<td>1</td>
</tr>
</tbody>
</table>
Keybag unlock

Passcode "1234"

Keybag SALT
Keybag ITER

Wrapped class key (WPKY)

WRAP = 3 (2\&1)
WRAP = 1

UID key

Key 0x835

Passcode key

KDF

AES unwrap

integrity check fail => wrong passcode

AES decrypt

Class key

iPhone data protection in depth
Escrow Keybags

Definition

- Copy of the system keybag, protected with random 32 byte passcode
- Stored off-device
- Escrow keybags passcodes stored on device
  - /private/var/root/Library/Lockdown/escrow_records

Usage

- iTunes, allows backup and synchronization without entering passcode
  - Device must have been paired (plugged in while unlocked) once
  - Stored in %ALLUSERSPROFILE%\Apple\Lockdown
- Mobile Device Management
  - Sent to MDM server during check-in, allows remote passcode change
Keychain

Description

- SQLite database (keychain-2.db)
- 4 tables: genp, inet, cert, keys
- securityd daemon handles database access
- Keychain API: IPC calls to securityd
- Access control: access group from caller’s entitlements (application identifier)
  - WHERE agrp=... clause appended to SQL statements
- On iOS 4, applications can specify a protection class (kSecAttrAccessible***) for their secrets
  - Each protection class has a ThisDeviceOnly variant
- Secrets encrypted with unique key, wrapped by class key
Keychain

Data column format

- System Keybag
- kSecAttr*** class key
- Encrypted item
- Wrapped item key
- 0

AES Wrap

AES

iPhone data protection in depth
Keychain

### Protection for build-in applications items

<table>
<thead>
<tr>
<th>Item</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi passwords</td>
<td>Always</td>
</tr>
<tr>
<td>IMAP/POP/SMTP accounts</td>
<td>AfterFirstUnlock</td>
</tr>
<tr>
<td>Exchange accounts</td>
<td>Always</td>
</tr>
<tr>
<td>VPN</td>
<td>Always</td>
</tr>
<tr>
<td>LDAP/CalDAV/CardDAV accounts</td>
<td>Always</td>
</tr>
<tr>
<td>iTunes backup password</td>
<td>WhenUnlockedThisDeviceOnly</td>
</tr>
<tr>
<td>Device certificate &amp; private key</td>
<td>AlwaysThisDeviceOnly</td>
</tr>
</tbody>
</table>
# Keychain Viewer

## Description
- Graphical application for jailbroken devices
- Inspect Keychain items content and attributes
- Show items protection classes

## Implementation
- Access `keychain-2.db` directly (read only)
- Calls AppleKeyStore `KeyUnwrap` selector to get items keys
  - Requires `com.apple.keystore.access-keychain-keys` entitlement
- Has to run as root (source code available)
Passcode derivation

Description

- AppleKeyStore exposes methods to unlock keybags
  - UnlockDevice, KeyBagUnlock
- Passcode derivation is done in kernel mode
- Transforms user’s passcode into passcode key
- Uses hardware UID key to tie passcode key to the device
  - Makes bruteforce attacks less practical
- Resulting passcode key is used to unwrap class keys
  - If AES unwrap integrity check fails, then input passcode is wrong
- Bruteforce possible with unsigned code execution, just use the AppleKeyStore interface
Passcode derivation algorithm

**Initialization**

- $A = A_1 = \text{PBKDF2}(\text{passcode}, \text{salt}, \text{iter}=1, \text{outputLength}=32)$

**Derivation (390 iterations)**

- XOR expand $A$ to 4096 bytes
  - $B = A \oplus 1 | A \oplus 2 | \ldots$
  - Keybag V2: $B = A_1 \oplus \text{counter++} | A_1 \oplus \text{counter++} | \ldots$
- AES encrypt with hardware UID key
  - $C = \text{AES}_{\text{ENCRYPT}_{\text{UID}}}(B)$: must be done on the target device
  - Last encrypted block is reused as IV for next round
- XOR $A$ with AES output
  - $A = A \oplus C$
Bruteforce attack

Using MobileKeyBag framework

```c
// load and decrypt keybag payload from systembag.kb
CFDictionaryRef kbdict = AppleKeyStore_loadKeyBag("/mnt2/keybags", "systembag");

CFDataRef kbkeys = CFDictionaryGetValue(kbdict, CFSTR("KeyBagKeys"));

// load keybag blob into AppleKeyStore kernel module
AppleKeyStoreKeyBagCreateWithData(kbkeys, &keybag_id);
AppleKeyStoreKeyBagSetSystem(keybag_id);

CFDataRef data = CFDataCreateWithBytesNoCopy(0, passcode, 4, NULL);
for (i=0; i < 10000; i++)
{
    sprintf(passcode, "%04d", i);
    if (!MKBUnlockDevice(data))
    {
        printf("Found passcode: %s\n", passcode);
        break;
    }
}
```
Bruteforce attack

Bruteforce speed

<table>
<thead>
<tr>
<th>Device</th>
<th>Time to try 10000 passcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPad 1</td>
<td>~16min</td>
</tr>
<tr>
<td>iPhone 4</td>
<td>~20min</td>
</tr>
<tr>
<td>iPhone 3GS</td>
<td>~30min</td>
</tr>
</tbody>
</table>

Implementation details

- MobileKeyBag framework does not export all the required functions (AppleKeyStore***)
  - Easy to re-implement

- No passcode set: system keybag protected with empty passcode

- Passcode "keyboard complexity" stored in configuration file
  - /var/mobile/Library/ConfigurationProfiles/UserSettings.plist
Bruteforce attack - Custom ramdisk

**Ramdisk creation**
- Extract restore ramdisk from any 4.x ipsw
- Add msft_guy sshd package (ssh.tar)
- Add bruteforce/key extractor tools

**Ramdisk bootstrap**
- Chronic dev team syringe injection tool (DFU mode exploits)
- Minimal cyanide payload patches kernel before booting
  - Patch IOAESAccelerator kext to allow UID key usage
  - Once passcode is found we can compute the passcode key from userland
- Same payload and ramdisk works on all A4 devices and iPhone 3GS
Bruteforce attack - Ramdisk tools

**Custom restored daemon**

- Initializes usbmux, disables watchdog
- Forks sshd
- Small plist-based RPC server
- Python scripts communicate with server over usbmux
- Plist output
Bruteforce attack - Ramdisk tools

**Bruteforce**

- Decrypt system keybag binary blob
- Load in AppleKeyStore kernel extension
- Try all 4-digit passcodes, if bruteforce succeeds:
  - Passcode, Passcode key (derivation function reimplemented)
  - Unwrapped class keys
  - Keychain can be decrypted offline
  - Protected files access through modified HFSExplorer
  - In-kernel keybag unlocked, protected files can also be retrieved directly using scp or sftp

**Escrow keybags**

- Get escrow keybag passcode from device
- Compute passcode key
- Get class keys without bruteforce
Plan

1 Introduction

2 Data protection

3 Storage encryption
   Introduction
   Effaceable area
   HFS Content Protection
   HFSExpplorer
   Data Wipe

4 iTunes Backups

5 Conclusion
iPhone storage

Introduction

- iPhone 3GS and below use NOR + NAND memory
- Newer devices only use NAND (except iPad 1)
- NAND encryption done by DMA controller (CDMA)
- Software Flash Translation Layer (FTL)
  - Bad block management, wear levelling
  - Only applies to filesystem area

NAND terminology

- Page: read/write unit
- Block: erase unit
Filesystem encryption

**Algorithm**

- AES in CBC mode
- Initialization vector depends on logical block number
- Hardcoded key for system partition (f65dae950e906c42b254cc58fc78eece)
- 256 bit key for data partition (EMF key)

**IV computation**

```c
void iv_for_lbn(unsigned long lbn, unsigned long *iv)
{
    for(int i = 0; i < 4; i++)
    {
        if(lbn & 1)
            lbn = 0x80000061 ^ (lbn >> 1);
        else
            lbn = lbn >> 1;
        iv[i] = lbn;
    }
}
```

iPhone data protection in depth
## Data partition encryption

### iOS 3
- MBR partition type 0xAE (Apple_Encrypted)
- EMF key stored in data partition last logical block
- Encrypted with key 0x89B

### iOS 4
- GPT partition table, EMF GUID
- EMF key stored in effaceable area
- Encrypted with key 0x89B
- HFS content protection
Data partition encryption - iOS 3

Encrypted key format

```c
struct crpt_ios3 {
    uint32_t magic0; // 'tprc'
    struct encrypted_data // encrypted with key89b CBC mode zero iv {
        uint32_t magic1; // 'TPRC'
        uint64_t partition_last_lba; // end of data partition
        uint32_t unknown; // 0xFFFFFFFF
        uint8_t filesystem_key [32]; // EMF key
        uint32_t key_length; // =32
        uint32_t pad_zero [3];
    }
};
```
iOS 4 NAND layout

**Container partitions**

- **boot**: Low Level Bootloader (LLB) image
- **plog**: Effaceable area
- **nvrm**: nvram, contains environments variables
- **firm**: iBoot, device tree, boot logos (IMG3 images)
- **fsys**: Filesystem partition, mapped as /dev/disk0

**16 Gb iPhone 4 NAND layout**

| boot block 0 | plog block 1 | nvrm blocks 2 - 7 | firm blocks 8 - 15 | fsys blocks 16 - 4084 | reserved blocks 4085 - 4100 |

- 4 banks of 4100 blocks of 128 pages of 8192 bytes data, 448 bytes spare
iOS 4 Storage encryption overview

- **System Keybag**: NSFileProtectionComplete class key
- **Passcode key**: Passcode
- **UID key**: key 0x89B, key 0x835
- **MKBPayload**: NAND key
- **EMF key**: Dkey, BAG1 key
- **GPT**: System partition, Data partition, systembag.kb (NSFileProtectionNone)
- **File contents**: (NSFileProtectionComplete)
- **Passcode derivation function**
- **Passcode**: AES Wrap

---

iPhone data protection in depth
Effaceable area

Plog partition

- Stores small binary blobs ("lockers")
- Abstract AppleEffaceableStorage kernel service
- Two implementations: AppleEffaceableNAND, AppleEffaceableNOR
- AppleEffaceableStorage organizes storage in groups and units
- For AppleEffaceableNAND, 4 groups (1 block in each bank) of 96 units (pages)
Effaceable area
Plog structures

### Plog Unit Header
- header[0:16] XOR header[16:31] = 'ecaF' + 0x1 + 0x1 + 0x0
- generation: incremented at each write
- crc32 (headers + data)

### Plog lockers format
```
kL  length  locker tag  locker data
```
Effaceable lockers

**EMF!**
- Data partition encryption key, encrypted with key 0x89B
- Format: length (0x20) + AES(key89B, emfkey)

**Dkey**
- NSProtectionNone class key, wrapped with key 0x835
- Format: AESWRAP(key835, Dkey)

**BAG1**
- System keybag payload key
- Format: magic (BAG1) + IV + Key
- Read from userland by keybagd to decrypt systembag.kb
- Erased at each passcode change to prevent attacks on previous keybag
AppleEffaceableStorage

### AppleEffaceableStorage IOKit userland interface

<table>
<thead>
<tr>
<th>Selector</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>getCapacity</td>
<td>960 bytes</td>
</tr>
<tr>
<td>1</td>
<td>getBytes</td>
<td>requires PE_i_can_has_debugger</td>
</tr>
<tr>
<td>2</td>
<td>setBytes</td>
<td>requires PE_i_can_has_debugger</td>
</tr>
<tr>
<td>3</td>
<td>isFormatted</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>format</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>getLocker</td>
<td>input : locker tag, output : data</td>
</tr>
<tr>
<td>6</td>
<td>setLocker</td>
<td>input : locker tag, data</td>
</tr>
<tr>
<td>7</td>
<td>effaceLocker</td>
<td>scalar input : locker tag</td>
</tr>
<tr>
<td>8</td>
<td>lockerSpace</td>
<td>?</td>
</tr>
</tbody>
</table>
HFS Content Protection

Description

• Each file data fork is encrypted with a unique file key
• File key is wrapped and stored in an extended attribute
  - com.apple.system.cprotect
• File protection set through F_SETPROTECTIONCLASS fcntl
• Some headers appear in the opensource kernel

Protection for build-in applications files

<table>
<thead>
<tr>
<th>Files</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mails &amp; attachments</td>
<td>NSProtectionComplete</td>
</tr>
<tr>
<td>Minimized applications screenshots</td>
<td>NSProtectionComplete</td>
</tr>
<tr>
<td>Everything else</td>
<td>NSProtectionNone</td>
</tr>
</tbody>
</table>
HFS Content Protection

cprotect extended attribute format

```
struct cprotect_xattr {
    uint16_t xattr_version; // =2 (version?)
    uint16_t zero; // =0
    uint32_t unknown; // leaks stack dword in one code path :)
    uint32_t protection_class_id;
    uint32_t wrapped_length; // 40 bytes (32 + 8 bytes from
                               // aes wrap integrity)
    uint8_t wrapped_key[1]; // wrapped_length
};
```
HFSExplorer

Motivation

- Standard dd image of iOS 4 data partition yields unreadable files
- When reading data partition from block device interface, each block is decrypted using the EMF key
  - Files data forks decrypted incorrectly

HFSExplorer additions

- Support for inline extended attributes
- Reads EMF, Dkey and other class keys from plist file
- Unwraps cprotect attributes to get file keys
- For each block in data fork:
  - Encrypt with EMF key to get original ciphertext
  - Decrypt with file key
  - (HFS allocation block size == NAND page size)
Data Wipe

Trigger

- Preferences → General → Reset → Erase All Content and Settings
- Erase data after $n$ invalid passcode attempts
- Restore firmware
- MobileMe Find My iPhone
- Exchange ActiveSync
- Mobile Device Management (MDM) server
Data Wipe

**Operation**

- `mobile_obliterator` daemon
- Erase DKey by calling `MKBDeviceObliterateClassDKey`
- Erase EMF key by calling selector `0x14C39` in `EffacingMediaFilter` service
- Reformat data partition
- Generate new system keybag
- High level of confidence that erased data cannot be recovered
iOS 4 Data wipe

- Introduction
- Data protection
- Storage encryption
- iTunes Backups
- HFS Content Protection
- HFSEncryptor
- Data Wipe

**iPhone data protection in depth**
Plan

1 Introduction

2 Data protection

3 Storage encryption

4 iTunes Backups
   Files format
   Keybag format
   Keychain format
   iTunes backup decrypter

5 Conclusion
Backed up files

Backup storage

- One directory per backup
- `%APPDATA%/Apple Computer/MobileSync/Backup/<udid>`
- **Can be password protected**
- Each file stored in a separate file
  - Encrypted (AES-256 CBC)
  - Filenames: SHA1 hashes

Database: MBDB

- Custom format
- Two files: Manifest.mbdb, Manifest.mbdx
- Contains information to restore files correctly
  - Filenames, size, permissions, extended attributes, etc.
### Files format
- Keybag format
- Keychain format
- iTunes backup decrypter

### iPhone data protection in depth

<table>
<thead>
<tr>
<th>Files format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info.plist</td>
<td>Manifest.mbdb</td>
</tr>
<tr>
<td>86736007d0166a18c646c567279b75093fc066fe</td>
<td>a690d7769cce8904ca2b67320b107c8fe5f79412</td>
</tr>
</tbody>
</table>

- d1f062e2da26192a6625d968274bfda8d07821e4
- d29f4fba1c2a95d92b05d53c1b9c967df6e02d5
- d351344f01cbe4900c9e981d1fb7ea5614e7c2e5
- e1cf61027554a85729b42484d8a33103f0317e26
- e452abcd1c5829fc16884318df4b8b14d3532a2
Database format

**mbdx = index**
- hex filenames
- file information offset in mbdb

**mbdb = data**
- Sequence of MBFileRecord
- Path, digest, etc.
- Encryption key, different for each file
  - ...and wrapped by class keys from backup keybag
## iPhone data protection in depth

### Manifest.mbdx

<table>
<thead>
<tr>
<th>Number of entries</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbdx...</td>
<td>...R_f/</td>
</tr>
<tr>
<td>...9UN...</td>
<td>.cpn...</td>
</tr>
<tr>
<td>UASS/Å...</td>
<td>.</td>
</tr>
<tr>
<td>j:CSCü...</td>
<td>Q.Älj</td>
</tr>
<tr>
<td>...XFZÊN)ö!}ø.</td>
<td>=</td>
</tr>
<tr>
<td>7å.ŞW Eq.</td>
<td>.z.</td>
</tr>
<tr>
<td>L√5Ö,</td>
<td>^ g. t. E-</td>
</tr>
<tr>
<td>/±Mä,</td>
<td>Ėëü, vU</td>
</tr>
<tr>
<td>.rz~Ü...PAqYs</td>
<td>i</td>
</tr>
<tr>
<td>ŒĐyD%</td>
<td>...Z&lt;mI</td>
</tr>
<tr>
<td>A℮hA`</td>
<td>...W.6v</td>
</tr>
</tbody>
</table>

### Manifest.mbdb

<table>
<thead>
<tr>
<th>MBFileRecord entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001F40 EE 4D F1 D2 EE 00 00 00 00 50 04 00 00</td>
</tr>
<tr>
<td>000001F50 0A 48 6F D1 6D 65 44 6F 6D 61 69 64 EE 00 00</td>
</tr>
<tr>
<td>000001F60 72 61 72 79 2F 50 72 65 66 65 72 65 6E 63 69</td>
</tr>
<tr>
<td>000001F70 63 77 6E 61 69 6E 63 69 73 74 2E 70 6F 62 61</td>
</tr>
<tr>
<td>000001F80 63 6E 61 69 6E 63 69 73 74 2E 70 6F 62 61 69</td>
</tr>
<tr>
<td>000001F90 15 35 D8 2E 6D 02 56</td>
</tr>
<tr>
<td>000001FA0 7C 92 5E 6E 64 6F 62 74 65 2E 69 6D 61 69 64</td>
</tr>
<tr>
<td>000001FB0 94 AA 86 12 37 84 74 63 76 8A 32 97 C5 91 7D</td>
</tr>
<tr>
<td>000001FC0 54 4A 5D 6D C5 E4 98 83 86 85 28 D0 5F 8C E6</td>
</tr>
<tr>
<td>000001FD0 0D 47 81 80 00 00 00 00 00 00 00 00 00 00 00 01</td>
</tr>
<tr>
<td>000001FE0 00 00 01 F5 4D D3 A1 27 4D D3 A1 27 4D D3 A1 27</td>
</tr>
<tr>
<td>000001FF0 00 00 00 00 00 00 00 00 01 86 04 00 00 00 0A 48</td>
</tr>
</tbody>
</table>

### Files format

- **Keybag format**
- **Keychain format**
- **iTunes backup decrypter**

### Database format

<table>
<thead>
<tr>
<th>Number of entries</th>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000</td>
<td>6D 62 64 78 02 00 00 00 00 9D</td>
</tr>
<tr>
<td>000000010</td>
<td>9E BA 39 86 84 AF B6 9B A5 03 A2 70 96 67</td>
</tr>
<tr>
<td>000000020</td>
<td>1F 49 81 80 E7 53 2F 80 8C 1E 24 E4 BF 0B 06 81</td>
</tr>
<tr>
<td>000000030</td>
<td>6A D4 3B 43 B7 D7 9F 95 00 00 51 4F 81 80 6C 6A</td>
</tr>
<tr>
<td>000000040</td>
<td>11 06 1D 58 46 5A E6 84 29 B2 9B 21 7D BF 14 3D</td>
</tr>
<tr>
<td>000000050</td>
<td>1C D0 00 00 37 8B 81 A4 57 AB E9 71 89 04 7A 81</td>
</tr>
<tr>
<td>000000060</td>
<td>4C C3 35 CD E2 D7 20 F6 19 67</td>
</tr>
<tr>
<td>000000070</td>
<td>81 B6 2F D6 4D 8A AF FC DB E9 B0 9F CD FC 76</td>
</tr>
<tr>
<td>000000080</td>
<td>0B 5C 72 7A F7 F3 00 00 07 50 41 C0 71 B4 73 93</td>
</tr>
<tr>
<td>000000090</td>
<td>F1 45 C6 D8 44 A8 E4 F8 95 15 08 5A DC D3 6D</td>
</tr>
<tr>
<td>0000000A0</td>
<td>00 00 07 41 C0 BE DE C6 D4 2E FE 57 12 36 16</td>
</tr>
</tbody>
</table>

### Database entries

<table>
<thead>
<tr>
<th>MBFileRecord entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000</td>
</tr>
<tr>
<td>000000010</td>
</tr>
<tr>
<td>000000020</td>
</tr>
<tr>
<td>000000030</td>
</tr>
<tr>
<td>000000040</td>
</tr>
<tr>
<td>000000050</td>
</tr>
<tr>
<td>000000060</td>
</tr>
<tr>
<td>000000070</td>
</tr>
<tr>
<td>000000080</td>
</tr>
<tr>
<td>000000090</td>
</tr>
<tr>
<td>0000000A0</td>
</tr>
</tbody>
</table>
Backup keybag

- Same format as before
- Stored in Manifest.plist
  - BackupKeyBag section
- Random class keys for each backup
  - Different from system keybag keys

Not all the keys can be retrieved
Backup keychain

- Stored in keychain-backup.plist
- Same structure as keychain-2.db, but in a plist
- Before accessing it:
  - Backup needs to be decrypted
  - Filenames need to be recovered
- Decrypt items using keychain class keys from backup keybag
iTunes backup decrypter

Requirements

- Needs password if protected
- Wrote a bruteforcer (slow)

Implementation

- Decrypted files in a new directory
- Filenames can be restored or not
- MBFileRecord fully documented
- Integrated keychain viewer
Plan

1 Introduction
2 Data protection
3 Storage encryption
4 iTunes Backups
5 Conclusion
Conclusion

Data protection

- Significant improvement over iOS 3
- Derivation algorithm uses hardware key to prevent attacks
- Bruteforce attack only possible due to BootROM vulnerabilities
- Only Mail files are protected by passcode
  - Should be adopted by other build-in apps (Photos, etc.)
  - Might be difficult in some cases (SMS database)

Tools & Source code

- [http://code.google.com/p/iphone-dataprotection/](http://code.google.com/p/iphone-dataprotection/)
Thank you for your attention

Questions ?
References

- Apple WWDC 2010, Session 209 - Securing Application Data
- The iPhone wiki, http://www.theiphonewiki.com
- msftguy ssh ramdisk http://msftguy.blogspot.com/
- AES wrap, RFC 3394 http://www.ietf.org/rfc/rfc3394.txt
- NAND layout, CPICH
- HFSExplorer, Erik Larsson http://www.catacombae.org/hfsx.html
- syringe, Chronic dev team https://github.com/Chronic-Dev/syringe
- cyanide, Chronic dev team https://github.com/Chronic-Dev/cyanide
- usbmux enable code, comex
  https://github.com/comex/bloggy/wiki/Redsn0w%2Busbmux
- restored_pwn, Gojohnnyboi
  https://github.com/Gojohnnyboi/restored_pwn
References

- xpwn crypto tool, planetbeing https://github.com/planetbeing/xpwn
- iPhone backup browser
  http://code.google.com/p/iphonebackupbrowser/