iPhone data protection in depth

Jean-Baptiste Bédrune
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 vulnerablities

- 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)

iPhone crypto

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)

iOS 4

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

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

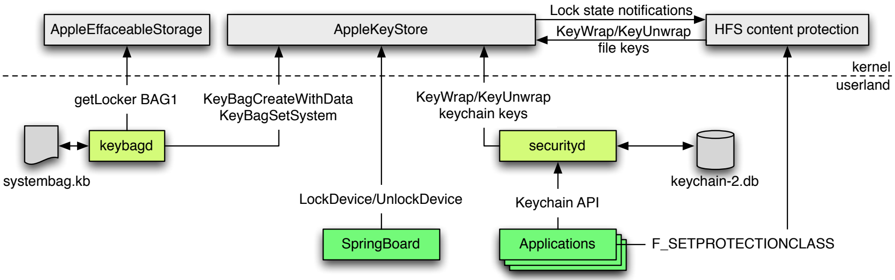
Protection classes

Availability	Filesystem	Keychain
When unlocked	NSProtectionComplete	WhenUnlocked
After first unlock		AfterFirstUnlock
Always	NSProtectionNone	Always

Implementation

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

Data protection components & interactions



Keybagd

Description

- System daemon, loads system keybag into AppleKeyStore kernel service at boot
- Handles system keybag persistance 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

Example keybag hexdump

```
0000000: 4441 5441 0000 0444 5645 5253 0000 0004  DATA...DVERS....
0000010: 0000 0002 5459 5045 0000 0004 0000 0000  ....TYPE........
0000020: 5555 4944 0000 0010 ceea c20d cf52 40e0  UUID.........R@.
0000030: ac0e dd52 915d 38bc 484d 434b 0000 0028  ...R.]8.HMCK...(
0000040: 6785 4e94 bc50 f2e4 541b c51d 8f46 ad59  g.N..P..T....F.Y
0000050: 3af3 cdcb 201a 2e53 6424 b728 3775 788f  :... ..Sd$.(7ux.
0000060: cd2e 28f8 b692 2bac 5752 4150 0000 0004  ..(...+.WRAP....
0000070: 0000 0001 5341 4c54 0000 0014 8bda 11d7  ....SALT........
0000080: 43bb 669c e451 646c 2ea9 ac0b 6658 ff9d  C.f..Qdl....fX..
0000090: 4954 4552 0000 0004 0000 c350 5555 4944  ITER.......PUUID
00000a0: 0000 0010 02ed b2ea c187 49b2 b9f1 7925  ..........I...y%
00000b0: ddaa daae 434c 4153 0000 0004 0000 000b  ....CLAS........
00000c0: 5752 4150 0000 0004 0000 0001 5750 4b59  WRAP........WPKY
00000d0: 0000 0020 8f81 980c a483 2ae4 e978 4cc8  ... ......*..xL.
00000e0: f715 f4e3 44ac 71cc b568 22e6 e119 6983  ....D.q..h"...i.
00000f0: b156 e25e 5555 4944 0000 0010 d8e0 f7a2  .V.^UUID........
```

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 (?!)

Keybag binary format

Wrapped class keys

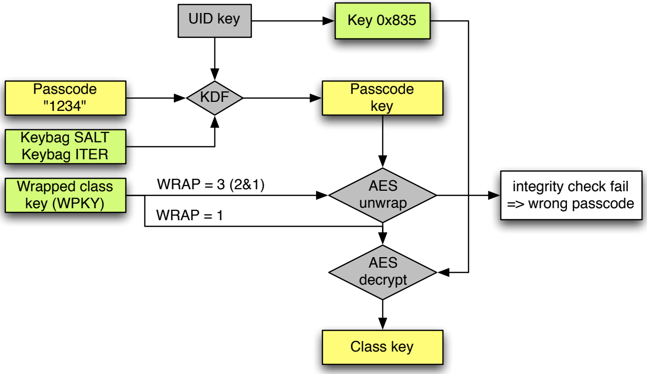
- `UUID` : Key uuid
- `CLAS` : Class number
- `WRAP` : Wrap flags
  - 1 : AES encrypted with key 0x835
  - 2 : AES wrapped with passcode key (RFC 3394)
- `WPKY` : Wrapped key

Class keys identifiers

Class keys

Id	Class name	Wrap
1	NSProtectionComplete	3
2	(NSFileProtectionWriteOnly)	3
3	(NSFileProtectionCompleteUntilUserAuthentication)	3
4	NSProtectionNone (stored in effaceable area)	x
5	unused ? (NSFileProtectionRecovery ?)	3
6	kSecAttrAccessibleWhenUnlocked	3
7	kSecAttrAccessibleAfterFirstUnlock	3
8	kSecAttrAccessibleAlways	1
9	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	3
10	kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly	3
11	kSecAttrAccessibleAlwaysThisDeviceOnly	1

Keybag unlock



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

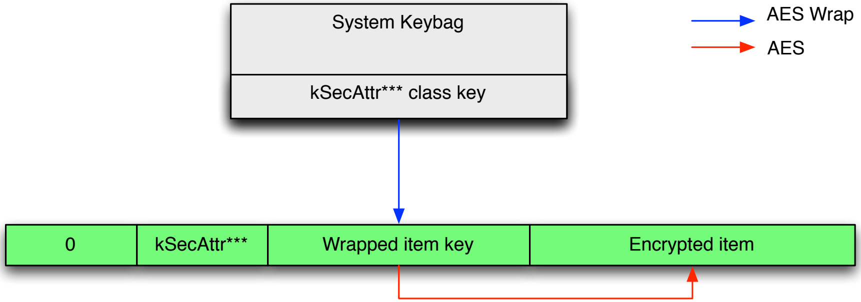
Keychain

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



Keychain

Protection for build-in applications items

Item	Accessibility
Wi-Fi passwords	Always
IMAP/POP/SMTP accounts	AfterFirstUnlock
Exchange accounts	Always
VPN	Always
LDAP/CalDAV/CardDAV accounts	Always
iTunes backup password	WhenUnlockedThisDeviceOnly
Device certificate & private key	AlwaysThisDeviceOnly

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 = A1 = PBKDF2(passcode, salt, iter=1, outputLength=32)

Derivation (390 iterations)

- XOR expand A to 4096 bytes
  - B = A $\oplus$ 1 | A $\oplus$ 2 | ...
  - Keybag V2 : B = A1 $\oplus$ counter++ | A1 $\oplus$ counter++ | ...
- AES encrypt with hardware UID key
  - $C = AES\_ENCRYPT\_UID(B)$: 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

```
//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

Device	Time to try 10000 passcodes
iPad 1	~16min
iPhone 4	~20min
iPhone 3GS	~30min

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 funtion 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

SOGETI

Plan

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

```
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;
    }
}
```

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

```
struct crpt_ios3
{
    uint32_t magic0; // 'tprc'

    struct encryted_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

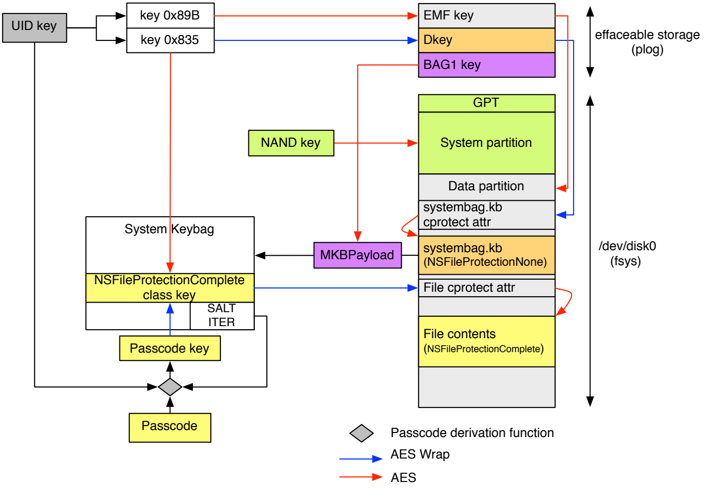
- boot : Low Level Bootloader (LLB) image
- plog : Effaceable area
- nvrm : nvram, contains environments variables
- firm : iBoot, device tree, boot logos (IMG3 images)
- fsys : Filesystem partition, mapped as /dev/disk0

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



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

```
0000000: f2db b184 3521 b498 602f 242c 8acb 41df ....5!..`/$,..A.
0000010: 97b8 d0c2 3421 b498 612f 242c 8acb 41df ....4!..a/$,..A.
0000020: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000030: 0000 0000 0000 0000 4900 0000 2b3d e1ad ........I...+=..
0000040: 6b4c 3400 3147 4142 3147 4142 ef3e 87cd kL4.1GAB1GAB.>..
0000050: 374b 39ef 68a0 8977 6ac5 b229 836e 758e 7K9.h..wj..).nu.
0000060: e1b2 d8a8 f14f 7203 933f 2552 1067 3804 .....Or..?%R.g8.
0000070: 4aaf f0dc d37e 6922 a17b 863b 6b4c 2800 J....~i".{.;kL(.
0000080: 7965 6bc4 63cc 890c 046e f855 3717 0284 yek.c....n.U7...
0000090: 5bfa c670 6ed9 e42b e0d5 58a7 b021 5b91 [..pn..+..X..![.
00000a0: 16d6 9de2 8333 02af e179 4416 6b4c 2400 .....3...yD.kL$.
00000b0: 2146 4dc5 2000 0000 9506 d2b1 5d48 df7f !FM. .......]H..
00000c0: 1fb2 ca2e 1aef cbff 8814 95f2 9e38 1ff1 .............8..
00000d0: ad4d 4484 8f38 50a5 6b4c 0000 454e 4f44 .MD..8P.kL..ENOD
```

Length **Tags**

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

Selector	Description	Comment
0	getCapacity	960 bytes
1	getBytes	requires `PE_i_can_has_debugger`
2	setBytes	requires `PE_i_can_has_debugger`
3	isFormatted	
4	format	
5	getLocker	input : locker tag, output : data
6	setLocker	input : locker tag, data
7	effaceLocker	scalar input : locker tag
8	lockerSpace	?

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
  - `http://opensource.apple.com/source/xnu/xnu-1504.9.37/bsd/sys/cprotect.h`

Protection for build-in applications files

Files	Accessibility
Mails & attachments	NSProtectionComplete
Minimized applications screenshots	NSProtectionComplete
Everything else	NSProtectionNone

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

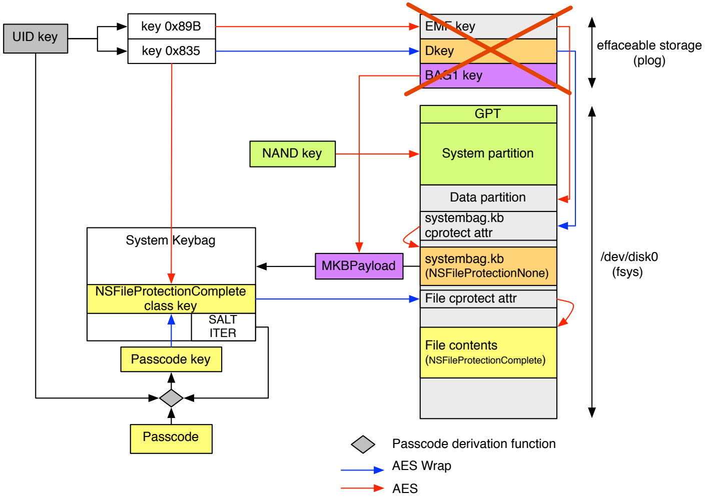
- Preferences → General → Reset → Erase All Content and Settings
- Erase data after *n* invalid passcode attempts
- Restore firmware
- MobileMe Find My iPhone
- Exchange ActiveSync
- Mobile Device Managment (MDM) server

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



Plan

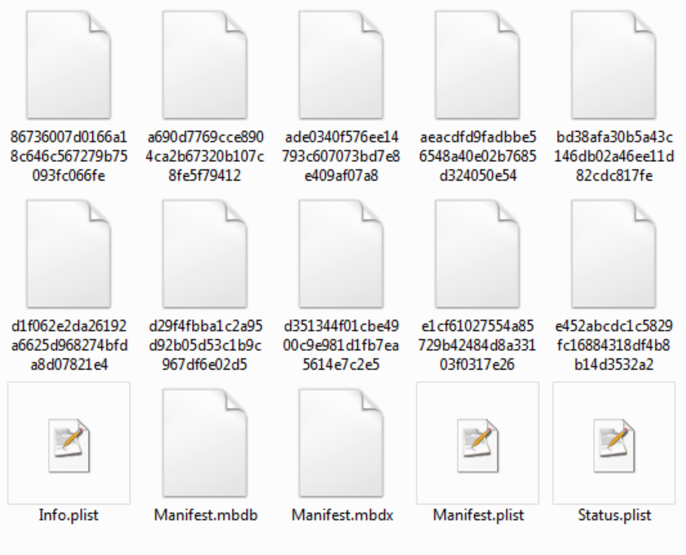
Backed up files

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.


86736007d0166a18c646c567279b75093fc066fe
a690d7769cce8904ca2b67320b107c8fe5f79412
ade0340f576ee14793c607073bd7e8e409af07a8
aeacdfd9fadbbe56548a40e02b7685d324050e54
bd38afa30b5a43c146db02a46ee11d82cdc817fe
d1f062e2da26192a6625d968274bfda8d07821e4
d29f4fbba1c2a95d92b05d53c1b9c967df6e02d5
d351344f01cbe4900c9e981d1fb7ea5614e7c2e5
e1cf61027554a85729b42484d8a33103f0317e26
e452abcdc1c5829fc16884318df4b8b14d3532a2
Info.plist
Manifest.mbdb
Manifest.mbdx
Manifest.plist
Status.plist

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

Database format

Number of entries

Filename

Manifest.mbdx

```
00000000  6D 62 64 78 02 00 00 00 00 9D 52 C0 3E DF C4 DA   mbdx......R¿>ﬂƒ/
00000010  9E BA 39 86 84 AF B6 9B A5 03 A2 70 96 67 00 00   û∫9ÜÑØ∂õ•.¢pñg..
00000020  1F 49 81 80 E7 53 2F 80 8C 1E 24 E4 BF 0B 06 81   .I.ÄÁS/Äå.$‰ø...
00000030  6A D4 3B 43 B7 D7 9F 50 00 00 51 4F 81 80 6C 6A   j‘;C∑◊üP..QO.Älj
00000040  11 06 1D 58 46 5A E6 84 29 B2 9B 21 7D BF 14 3D   ...XFZÊÑ)≤õ!}ø.=
00000050  1C D0 00 00 37 8B 81 A4 57 AB E9 71 89 04 7A 81   .–..7ã.§W´Èqâ.z.
00000060  4C C3 35 CD E2 D7 20 F6 19 67 2C 74 00 00 45 D1   L√5Õ‚◊ ˆ.g,t..E—
00000070  81 B6 2F D6 4D 8A AF FC DB E9 B0 9F CD FC 76 F4   .∂/÷MäØ˛€È∞üÕ˛vÙ
00000080  0B 5C 72 7A F7 F3 00 00 07 50 41 C0 71 B4 73 93   .\rz˜Û...PA¿q¥sì
00000090  F1 45 C6 D8 44 A8 E4 F8 95 15 08 5A DC D3 6D ED   ÒE∆ÿD®‰¯ï..Z‹”mÌ
000000A0  00 00 00 7F 41 C0 BE DE C6 D4 2E FE 57 12 36 76   ....A¿æﬁ∆‘.˛W.6v
```

Manifest.mbdb

```
00001F40  EE 4D D1 02 EE 00 00 00 00 00 00 00 50 04 00 00   ÓM—.Ó.......P...
00001F50  0A 48 6F 6D 65 44 6F 6D 61 69 6E 00 2F 4C 69 62   .HomeDomain./Lib
00001F60  72 61 72 79 2F 50 72 65 66 65 72 65 6E 63 65 73   rary/Preferences
00001F70  2F 63 6F 6D 2E 61 70 70 6C 65 2E 6D 6F 62 69 6C   /com.apple.mobil
00001F80  65 6E 6F 74 65 73 2E 70 6C 69 73 74 FF FF 00 14   enotes.plist˘˘..
00001F90  15 35 D8 [illegible] 6D 02 56   .5ÿUÀ=Õ<◊3+.Ûm.V
00001FA0  7C 92 5E [illegible] 84 28 45   |í^Ç.,.....©≤.Ñ(E
00001FB0  94 AA 86 12 37 84 74 C1 3F 76 8A 32 97 C5 91 7D   î™Ü.7Ñt¡?vä2ó≈ë}
00001FC0  54 4A 5D 6D C5 E4 98 83 86 85 28 D0 5F 8C E6 31   TJ]m≈‰òÉÜÖ(–_åÊ1
00001FD0  0D 47 81 80 00 00 00 00 00 00 59 63 00 00 01 F5   .G.Ä......Yc...ı
00001FE0  00 00 01 F5 4D D3 A1 27 4D D3 A1 27 4D D3 A1 27   ...ıM”°'M”°'M”°'
00001FF0  00 00 00 00 00 00 01 86 04 00 00 0A 48 6F 6D 65   .......Ü....Home
```

MBFileRecord entry

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

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/`

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
  http://theiphonewiki.com/wiki/index.php?title=NAND
- 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/