Forensic analysis of iPhone backups

The goal of iPhone Backup Forensics analysis is extracting data and artefacts from the iTunes backups without altering any information.

iPhone forensics can be performed on the backups made by iTunes or directly on the live device. My last article on iPhone forensics detailed the forensic techniques and the technical challenges involved in performing live device forensics. Forensic analysis on live device reboots the phone, may alter the information stored on the device. In critical investigations, forensic examiners rely on analyzing the iPhone logical backups acquired through iTunes. iTunes uses AFC (Apple file connection) protocol to take the backup and the backup process does not modify anything on the iPhone except the escrow key records. This article explains the technical procedure and challenges involved in extracting data and artefacts from the iPhone backups. Understanding the forensic techniques on iTunes backups is also useful in cases where we get physical access to the suspect's computer instead of the iPhone directly. When a computer is used to sync with the iPhone, most of the information on the iPhone is likely to be backed up onto the computer. So gaining access to the computer's file system will also gives access to the mobile devices' data.

Techniques explained in this article works on all Apple Devices which are running with iOS 5.

Note: iPhone 4 GSM model with iOS 5.0.1 is used for the demos. Backups shown in the article are captured on Mac OS X Lion 10.6 using iTunes 10.6.

Researchers at Sogeti Labs have released open source forensic tools (with the support of iOS 5) to read normal and encrypted iTunes backups. Below details outline their research and gives an overview on usage of backup recovery tools.

iOS Backups:

With iOS 5, data stored on the iPhone can be backed up to a computer with iTunes or to a cloud based storage with iCloud. The article briefs about iCloud backups and provides a deep analysis of iTunes backups.

iCloud Backup:

iCloud allows backup & restoring the iPhone contents over Wi-Fi/3G to a cloud with a registered Apple account. iCloud backups the photos, application data, device settings, messages and mail, etc. iCloud services were introduced to provide a computer free backup solution. It acts as a remote backup service and allows moving data seamlessly between different Apple devices like Mac, iPod and iPad. iCloud also provide services to track the lost phone, lock the device remotely and wipe the data remotely. iCloud limits the free backup storage to 5 Giga Bytes. However additional iCloud data storage can be purchased by paying annual fees to Apple. iCloud uses a secure token for authentication and secures the content by encrypting it when sent over the internet. Use of a secure

token for authentication eliminates the need to store iCloud password on devices. Apple also claims that, all the iCloud data except the emails and notes is stored encrypted on disk using 128 bit encryption algorithm. Encrypted data stored on the disk is decrypted on the fly when requested from an authentication device. Data stored on the iCloud can also be backed up to a computer. Detailed procedure is available at Apple documentation.

On the iPhone, iCloud backup storage can be turned on/off by navigating to *Settings -> iCloud -> Storage & Backup*.

iCloud Backup toggle is shown in Figure 1.

(Figure 1)

iCloud data is effectively safe from hackers as Apple provides the best authentication mechanism by enforcing the users to use strong passwords, which would prevent the brute force attacks. As long as the user uses a strong password, information stored on the iCloud is safe.

iTunes Backup:

iTunes is used to backup the iPhone to a computer. When the iPhone is connected to a computer for the first time and synced with iTunes, iTunes automatically creates a folder with device UDID (Unique device ID – 40 hexadecimal characters long) as the name and copies the device contents to the newly created folder. The iPhone can be synced with iTunes over Wi-Fi or over an USB connection. If the automatic sync option is turned off in iTunes, the user has to manually initiate the backup process whenever the device is connected to the computer. Once the backup folder is created on the computer, then each time when the device is synced with the iTunes, it will only update the files in the existing folder. During first sync iTunes takes a full backup of the device. From there on, iTunes only backup and overwrite the files which are modified on the device. The behaviour can be observed by looking at different timestamps for the files in the backup. iTunes also initiates an automated backup when the iPhone is updated or restored. During an iOS update/restore, iTunes creates a differential backup with a folder name [UDID] + '-' + [Time stamp] in the same backup location. iTunes backup location varies for different operating systems and the exact directory paths are listed in Table-1.

Backup files created by iTunes are platform independent and can be moved from one operating system to other.

Operating system	Backup Location
Windows XP	C:\Documents and Settings\[user name]\Application Data\Apple Computer\MobileSync\Backup\
MAC OS X	~/Library/Application Support/MobileSync/Backup/ (~ represents user's home directory)
Windows 7	C:\Users\[user name]\AppData\Roaming\Apple Computer\MobileSync\Backup\

(Table-1)

If a passcode protected iPhone is connected to the computer for the first time, iTunes will require the user to enter the passcode (shown in Figure 2) and unlock the device before starting the sync process.

(Figure 2)

Upon unlocking the iPhone with a valid passcode, iTunes recognizes the device as authorized and allows to backup and sync with the computer. From there on, iTunes will allow to backup or sync the iPhone without entering the passcode as long as it connects to the same computer. During backup, iTunes also creates a property list file with device UDID as the name and stores the Escrow key bag, Device certificate, Host ID, Host certificate and Host private key in it. Escrow Keybag allows a paired device (normally a computer) to gain full access to the iPhone file system (circumventing iOS Data Protection feature) when the phone is in a locked state. This improves the usability by not asking the user to unlock the device during every backup. Escrow key bag location varies for different operating systems and the exact directory paths are listed in Table-2.

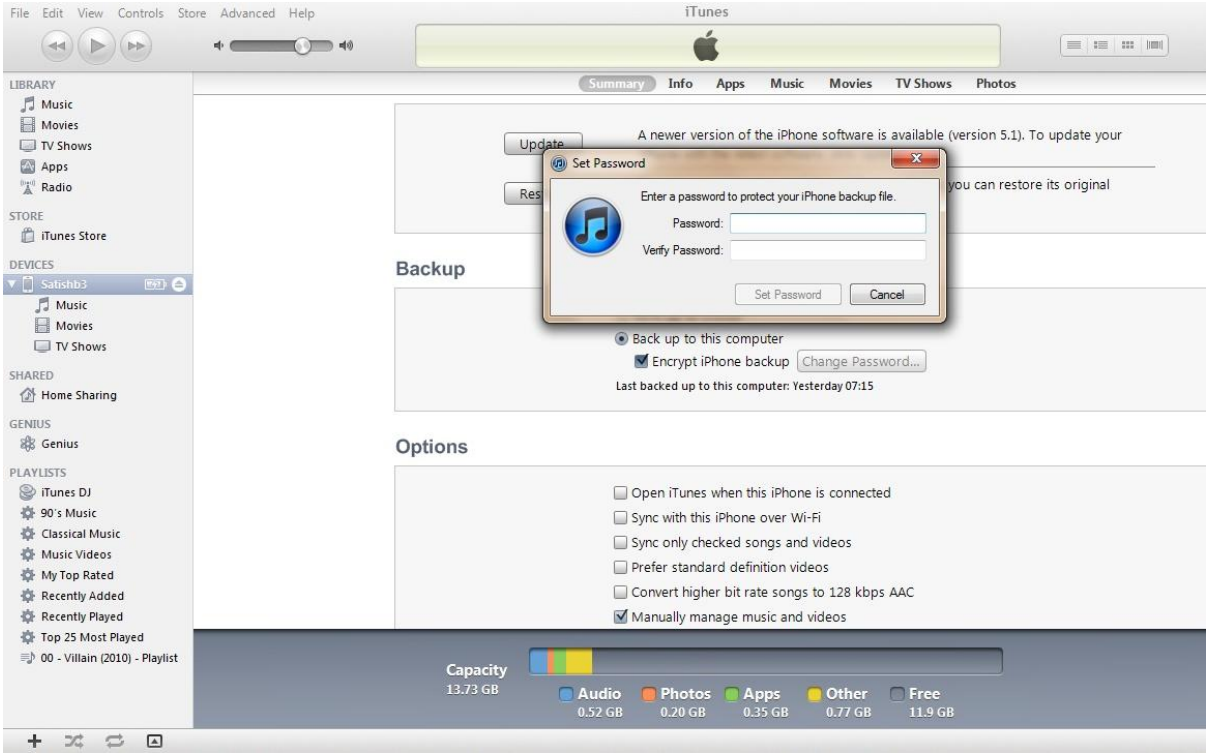
Operating system	Escrow keybag Location
Windows	%AllUsersProfile%\Apple\Lockdown\
MAC OS X	/private/var/db/lockdown/

(Table-2)

Escrow Keybag is encrypted with a key computed from the iPhone hardware (key 0x835) and it is protected with a 32 byte passcode which is stored on the iPhone. Escrow Keybag passcode gets stored in a PList file ([Host ID].plist) located at - */private/var/root/Library/Lockdown/escrow_records* directory on the iPhone. With iOS 5, Escrow Keybag is also protected with a passcode key derived from the user's passcode, restricting to perform Escrow Keybag attacks. Escrow Keybag attack bypasses the iPhone data protection mechanism and allows decrypting every file on the device without requiring the user's passcode. Escrow Keybag is a copy of the System Keybag and contains a collection of protection class keys that are used for data encryption on the iPhone. Protection class keys stored in the Escrow Keybag allows the iTunes to access protected files & keychain items when the iPhone is locked.

iTunes also creates a Backup Keybag for each backup. It consists of class keys that are different from the ones in the System Keybag. The files in the backup are encrypted using AES 256 in CBC mode, with a unique key and a null IV. These file keys are stored wrapped by a class key from the Backup Keybag. Keys in the Backup Keybag facilitate to store the backups in a secure manner. By default, Backup Keybag is encrypted with a key (key 0x835) derived from the iPhone hardware key (UID key). So even if someone gain access to the backup, it is not possible to retrieve all the data from the backup unless they know the hardware key, which can be achieved only through physical access to the device. As the backup files are encrypted with a hardware key, backup taken from a device can only be restored to the original device. With iOS 4, Apple introduced a feature to encrypt the iTunes backups, which provides portability and allows restoring the backup files of one device to another device. Encrypted backups are designed for data migration between different iOS devices. Data migration is achieved by encrypting the backup with a password that a user gives in iTunes instead of the devices hardware key. With encrypted backups, all the backup data can be migrated except the content which is protected by *ThisDeviceOnly* class keys.

To create encrypted backups, connect the device to the computer and select 'Encrypt iPhone Backup' option in iTunes. During the encrypted backup, iTunes prompt the user to enter a password as shown in the Figure 3. Later the password is used to encrypt all the files in the backup. iTunes also stores the backup password in iPhone keychain database. In encrypted backups, Backup Keybag is encrypted with the backup password. This would allow decrypting the backups without physical access to the device.

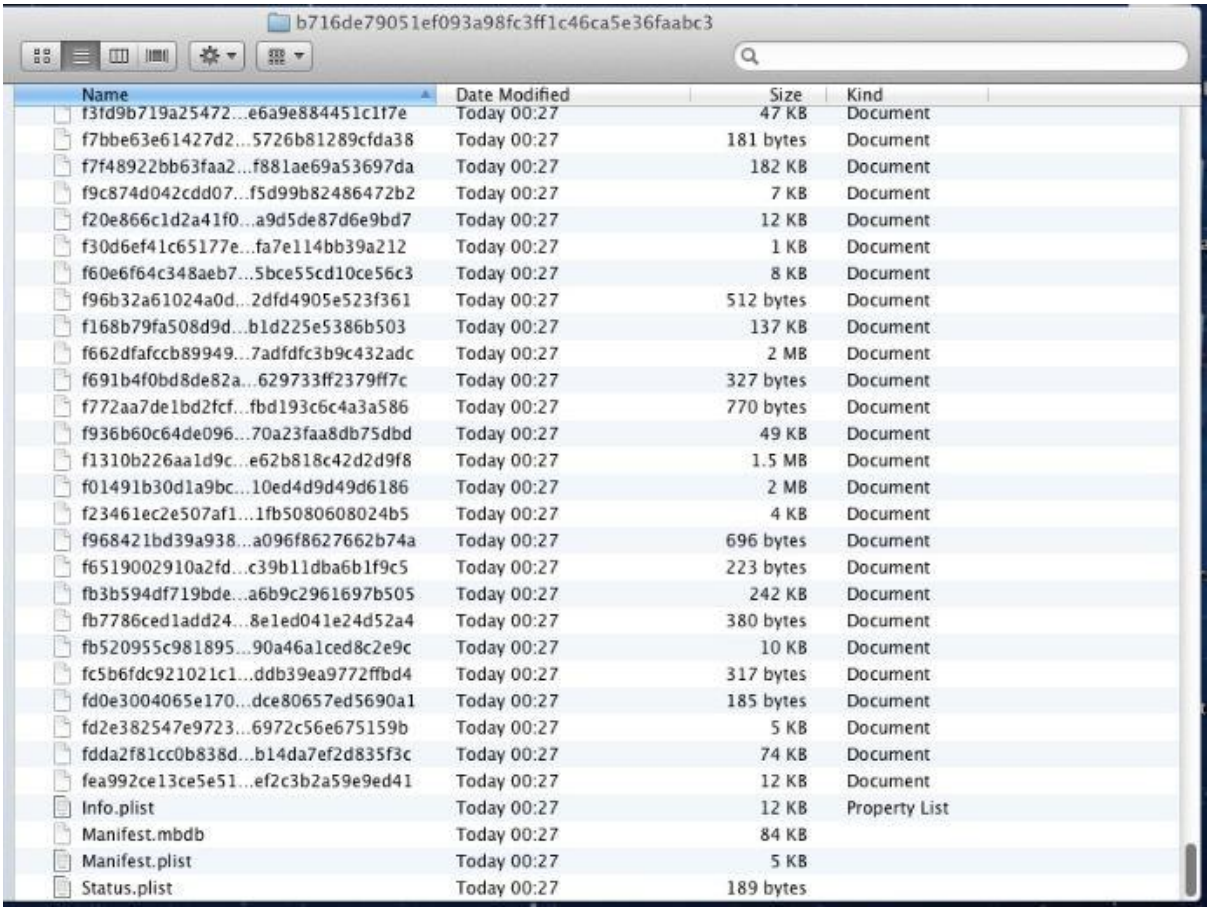


(Figure 3)

iTunes backup makes a copy of everything on the device like contacts, SMS, photos, calendar, music, call logs, configuration files, database files, keychain, network settings, offline web application cache, safari bookmarks, cookies and application data, etc. It also backups the device details like serial number, UDID, SIM hardware number and the phone number.

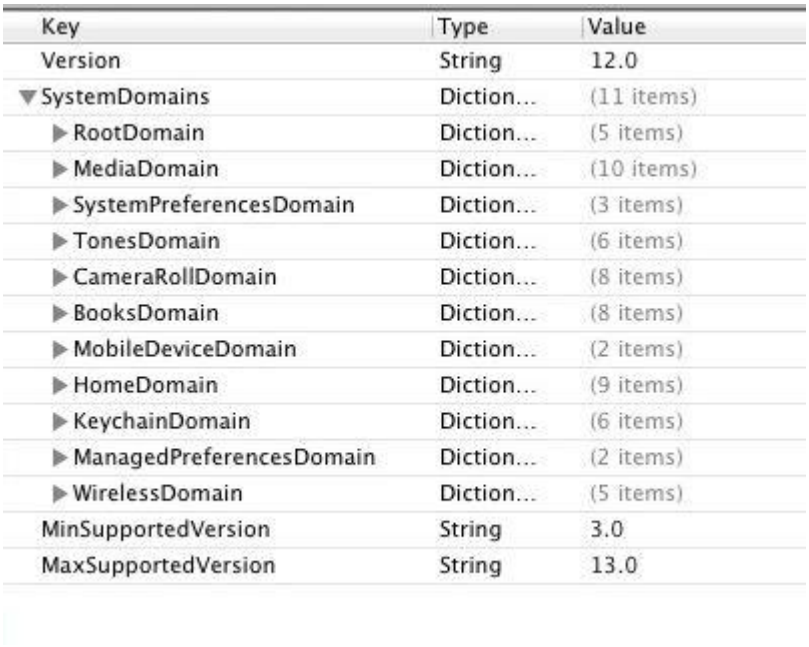
Backup folder contains a list of files which are not in a readable format and it consists of uniquely named files with a 40 digit alphanumeric hex value without any file extension. Example file name is: f968421bd39a938ba456ef7aa096f8627662b74a.

iTunes 10.6 backup of an iOS 5 device is shown in the Figure 4.



(Figure 4)

This 40 digit hex file name in the backup folder is the SHA1 hash value of the file path appended to the respective domain name with a '-' symbol. So the hash of *DomainName-filepath* will match to the correct file in the backup. In iOS 5, applications and inside data are classified into 12 domains (11 system domains and one application domain). The list of system domains can be viewed from */System/Library/Backup/Domains.plist* file on the iPhone. Domains.plist file content is shown in Figure 5.



Key	Type	Value
Version	String	12.0
▼ SystemDomains	Diction...	(11 items)
▶ RootDomain	Diction...	(5 items)
▶ MediaDomain	Diction...	(10 items)
▶ SystemPreferencesDomain	Diction...	(3 items)
▶ TonesDomain	Diction...	(6 items)
▶ CameraRollDomain	Diction...	(8 items)
▶ BooksDomain	Diction...	(8 items)
▶ MobileDeviceDomain	Diction...	(2 items)
▶ HomeDomain	Diction...	(9 items)
▶ KeychainDomain	Diction...	(6 items)
▶ ManagedPreferencesDomain	Diction...	(2 items)
▶ WirelessDomain	Diction...	(5 items)
MinSupportedVersion	String	3.0
MaxSupportedVersion	String	13.0

(Figure 5)

The method of managing the backups has changed with every major release of iTunes however the method of converting the path names to the file names still remains the same.

Few examples for path name to backup file name conversions are shown below -

Ex 1: Address book images backup file is - cd6702cea29fe89cf280a76794405adb17f9a0ee and this value is computed from SHA-1(*HomeDomain-Library/AddressBook/AddressBookImages.sqlitedb*).
**Online hash calculator - http://www.fileformat.info/tool/hash.htm?text=HomeDomain-Library%2FAddressBook%2FAddressBookImages.sqlitedb*

Ex 2: AppDomain is used for the applications which are downloaded from AppStore.
Skype property list backup file is - bc0e135b1c68521fa4710e3edadd6e74364fc50a and this value is computed from SHA-1(*AppDomain-com.skype.skype-Library/Preferences/com.skype.skype.plist*).
**Online Hash calculator - http://www.fileformat.info/tool/hash.htm?text=AppDomain-com.skype.skype-Library%2FPreferences%2Fcom.skype.skype.plist*

Ex 3: Keychain sqlite database backup file is - 51a4616e576dd33cd2abadfea874eb8ff246bf0e and this value is computed from SHA-1(*KeychainDomain-keychain-backup.plist*).
**Online Hash calculator - http://www.fileformat.info/tool/hash.htm?text=KeychainDomain-keychain-backup.plist*

iTunes stores/reads the domain names and path names from Meta files. Every iOS backup contains four Meta files - Info.plist, Manifest.plist, Status.plist and Manifest.mbdb along with the actual file contents.

Info.plist: The property list file contains the device details like device name, build version, IMEI, phone number, last backup date, product version, product type, serial number, sync settings and a list of application names that were installed on the device, etc.

Manifest.plist: The property list file contains the third party application bundle details, Backup Keybag, a flag to identify the passcode protected devices (WasPasscodeSet) and a flag to identify the encrypted backup (IsEncrypted), etc.

Status.plist: The property list file contains the details about the backup. It includes backup state, a flag to identify the full backup (IsFullBackup), date and version, etc.

Manifest.mbdb: The binary file contains information about all other files in the backup along with the file sizes and file system structure data. Backup file structure in older version of iTunes is managed by two files - Manifest.mbdx and Manifest.mbdb. In which, Manifest.mbdx file acts as an index file for the backup and indexes the elements that will be found in Manifest.mbdb. Since the introduction of iTunes 10, index file (mbdx) is eliminated and the backup is managed by a single mbdb file.

A sample Manifest.mbdb file is shown in Figure 6. As Manifest.mbdb is a binary file, a Hex editor is used to view the contents.

Manifest.mbdb

00000006	00 01 02 03	04 05	06 07	08 09	0a 0b	0c 0d	0e 0f	
00000006	6d 62 64 62	05 00	00 21	41 70	70 44	6f 6d	61 69	mbdb...!AppDomai
00000010	6e 2d 63 6f	6d 2e	61 74	6f 64	61 62	2e 66	61 73	n-com.atodab.fas
00000020	74 6c 61 6e	65 6c	69 74	65 00	00 ff	ff ff	ff ff	tlanelite..ÿÿÿÿÿ
00000030	ff 41 ed 00	00 00	00 00	00 24	af 00	00 01	f5 00	ÿAí.....$¯...õ.
00000040	00 01 f5 4f	4f 44	48 4f	4f 44	ad 4f	4f 44	48 00	..õOODHOOD-OODH.
00000050	00 00 00 00	00 00	00 04	01 00	1f 63	6f 6d	2e 61	.....com.a
00000060	70 70 6c 65	2e 69	6e 73	74 61	6c 6c	64 2e	73 74	pple.installd.st
00000070	61 74 69 63	5f 75	73 61	67 65	00 08	00 c0	ac 01	atic_usage...À¬.
00000080	00 00 00 00	00 21	41 70	70 44	6f 6d	61 69	6e 2d	.....!AppDomain-
00000090	63 6f 6d 2e	61 74	6f 64	61 62	2e 66	61 73	74 6c	com.atodab.fastl
000000a0	61 6e 65 6c	69 74	65 00	07 4c	69 62	72 61	72 79	anelite..Library
000000b0	ff ff ff ff	ff ff	41 ed	00 00	00 00	00 00	04 54	ÿÿÿÿÿÿAí.....T
000000c0	00 00 01 f5	00 00	01 f5	4f 4f	5e 21	4f 4f	5e 21	...õ...õOO^!OO^!
000000d0	4f 4d e5 2e	00 00	00 00	00 00	00 00	04 00	00 21	OMå.....!
000000e0	41 70 70 44	6f 6d	61 69	6e 2d	63 6f	6d 2e	61 74	AppDomain-com.at
000000f0	6f 64 61 62	2e 66	61 73	74 6c	61 6e	65 6c	69 74	odab.fastlanelit
00000100	65 00 0e 4c	69 62	72 61	72 79	2f 57	65 62	4b 69	e..Library/WebKi
00000110	74 ff ff ff	ff ff	ff 41	ed 00	00 00	00 00	00 3b	tÿÿÿÿÿÿAí.....;
00000120	00 00 00 01	f5 00	00 01	f5 4f	4f 5d	e7 4f	4f 5d	....õ...õOO]çOO]
00000130	e7 4f 4f 5d	e7 00	00 00	00 00	00 00	00 04	00 00	çOO]ç.....
00000140	21 41 70 70	44 6f	6d 61	69 6e	2d 63	6f 6d	2e 61	!AppDomain-com.a
00000150	74 6f 64 61	62 2e	66 61	73 74	6c 61	6e 65	6c 69	todab.fastlaneli
00000160	74 65 00 1b	4c 69	62 72	61 72	79 2f	57 65	62 4b	te..Library/WebK
00000170	69 74 2f 4c	6f 63	61 6c	53 74	6f 72	61 67	65 ff	it/LocalStorageÿ
00000180	ff ff ff ff	ff 41	ed 00	00 00	00 00	00 3b	01 00	ÿÿÿÿÿAí.....;..
00000190	00 01 f5 00	00 01	f5 4f	4f 5d	e7 4f	4f 5d	e7 4f	..õ...õOO]çOO]çO
000001a0	4f 5d e7 00	00 00	00 00	00 00	00 04	00 00	21 41	O]ç.....!A

(Figure 6)

Manifest.Mbdb file header and record format is shown in Table 3 & Table 4.

Header: Mbdb file header is a fixed value of 6 bytes and the value acts as a magic number to identify the mbdb files.

Type	Value
uint8[6]	mbdb\5\0

(Table 3)

Record: Mbdb file contain many records and each record is of variable size. Every record contains various details about a file.

Type	Data	Description
string	Domain	Domain Name
string	Path	File path
string	Target	Absolute path for Symbolic Links
string	Digest	SHA 1 hash Mostly None (0xff 0xff) for directories & AppDomain files 0x00 0x14 for System domain files
string	Encryption_key	None (0xff 0xff) for un encrypted files
uint16	Mode	Identifies the File Type '0xa000' for a symbolic link '0x4000' for a directory '0x8000' for a regular file
uint64	inode number	Lookup entry in inode table
uint32	User ID	Mostly 501
unit32	Group ID	Mostly 501
uint32	Last modified time	File last modified time in Epoch format
uint32	Last accessed time	File last accessed time in Epoch format
uint32	Created time	File created time in Epoch format
uint64	Size	Length of the file '0' for a symbolic link '0' for a directory Non zero for a regular file
uint8	Protection class	Data protection class (values 0x1 to 0xB)
uint8	Number of properties	Number of properties

(Table 4)

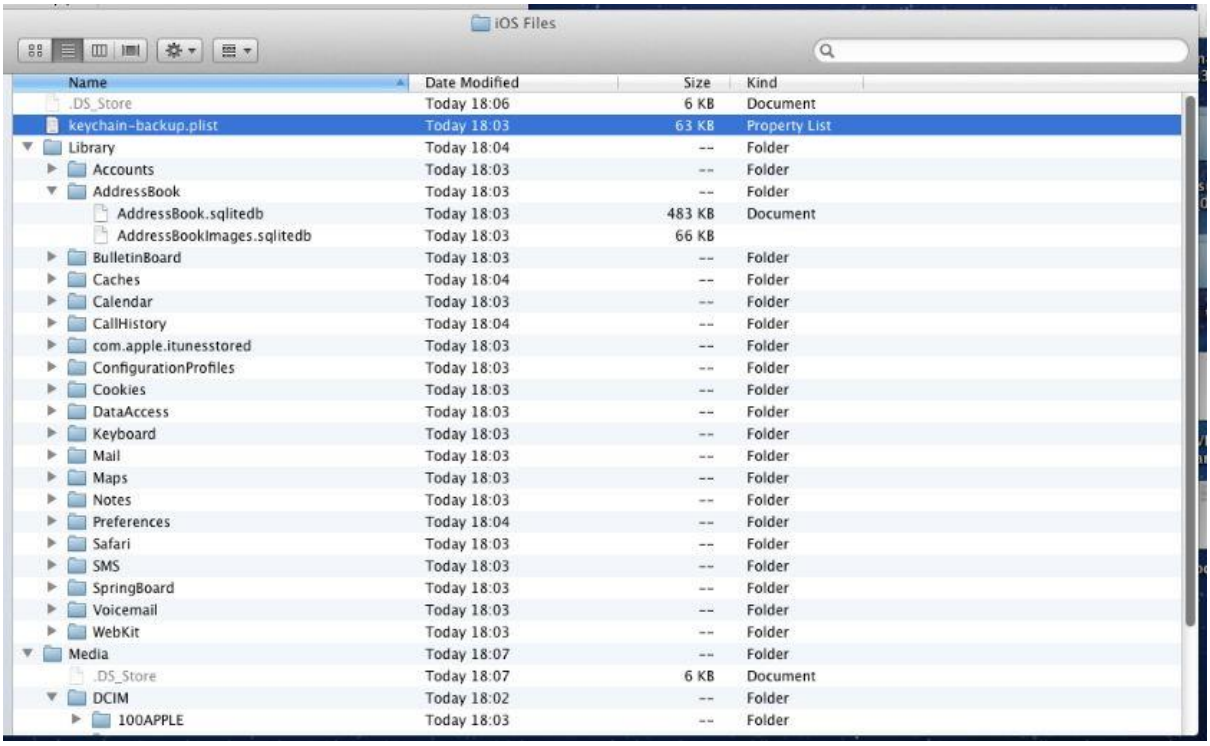
In the backup, most of the information is stored as plist files, sqlite database files and images files. Backup files can be viewed directly by adding an appropriate file extension.

Ex: Adding .plist file extension to bc0e135b1c68521fa4710e3edadd6e74364fc50a file allows viewing the contents of Skype property list file using a plist editor.

There are many free tools available to read iTunes backups. Some of the famous tools are listed here.

MAC OS X - iPhone Backup Extractor - http://supercrazyawesome.com/
Windows – iPhone Backup Browser - http://code.google.com/p/iphonebackupbrowser/
Mac OS X & Windows – iBackupBot - http://www.icopybot.com/itunes-backup-manager.htm

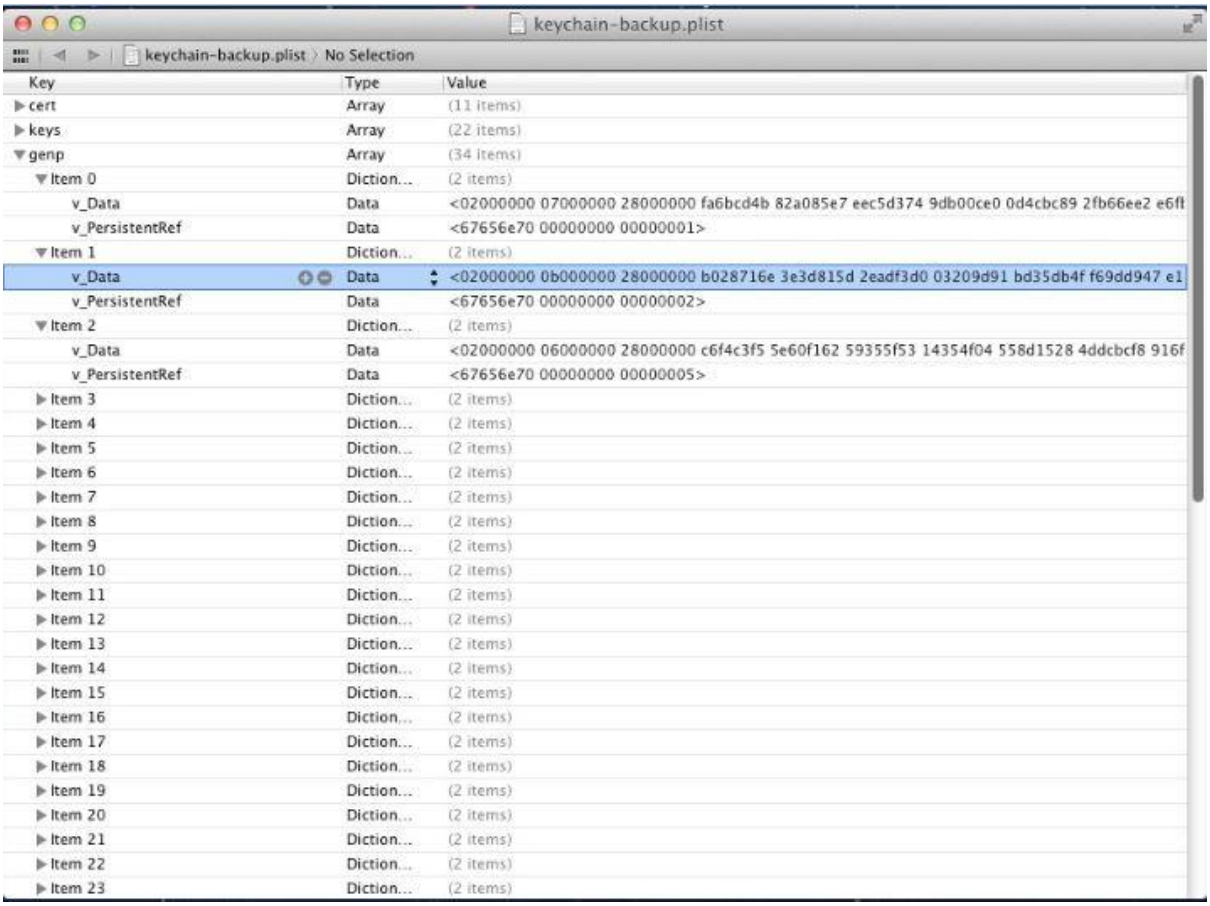
These tools parse the information stored in the Mbdb file and create the file structure. The tools convert the gibberish backup files into a readable format as shown in Figure 7.



(Figure 7)

Some of these tools leverage the Apple mobile devices API that comes with iTunes to create and read backups. The amount of information that can be extracted by the backup extractors is limited as the protected files in the backup are encrypted.

Ex: Keychain-backup.plist file extracted from the backup can be opened using a plist editor. However the contents inside the file are encrypted as shown in Figure 8.



(Figure 8)

Data protection mechanism introduced in iOS 4 protects the sensitive data in files on the file system and items in the keychain by adding another layer of encryption. Data protection uses the user's passcode key and the device specific hardware encryption keys to generate a set of class keys which protect the designated data. Developers use the data protection API to add protection class flag to the files and the keychain items. On the iPhone, protection class keys are stored in the System Keybag. During the backup, iTunes generates a new set of protection class keys and stores them in the Backup Keybag. Class keys stored in the System Keybag are different from the keys in the Backup Keybag. Protected files and data in the backup are encrypted using the class keys that are stored in the Backup Keybag. In normal backups Backup Keybag is protected with a key generated from the iPhone hardware (Key 0x835) and in encrypted backups it is protected with the iTunes password.

Data protection for files can be enabled by setting a value for the *NSFileProtection* attribute using the *NSFileManager* class *setAttributes:ofItemAtPath:error* method. List of protection classes available for the files are shown in Table 5.

Key id	Protection class	Description
1	NSProtectionComplete	File is accessible only after the device is unlocked
2	NSFileProtectionCompleteUnlessOpen	➢ File is accessible after the device is unlocked (or) ➢ File is accessible if the file handle remains open before locking the device
3	NSFileProtectionCompleteUntilFirstUserAuthentication	File is accessible after the first unlock of the device to till reboot
4	NSProtectionNone	File is accessible even the device is locked
5	NSFileProtectionRecovery	Undocumented

(Table 5)

Data protection for keychain items can be enabled by setting a protection class value in *SecItemAdd* or *SecItemUpdate* methods. Keychain class keys also define whether a keychain item can be migrated to other device or not. List of protection classes available for the keychain items are shown in Table 6

Key id	Protection class	Description
6	kSecAttrAccessibleWhenUnlocked	Keychain item is accessible only after the device is unlocked
7	kSecAttrAccessibleAfterFirstUnlock	Keychain item is accessible only after the first unlock of the device to till reboot
8	kSecAttrAccessibleAlways	Keychain item is accessible even the device is locked
9	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	Keychain item is accessible only after the device is unlocked and the item cannot be migrated between devices
10	kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly	Keychain item is accessible after the first unlock of the device and the item cannot be migrated
11	kSecAttrAccessibleAlwaysThisDeviceOnly	Keychain item is accessible even the device is locked and the item cannot be migrated

(Table 6)

Jean Sigwald, a researcher at Sogeti ESEC labs has released open source forensic tool kit that can be used to decrypt the protected backup files from normal backups and encrypted backups. Below details outline their research and gives an overview on usage of the tools.

Setup:

On Mac OS X, download & install the required python modules (*pycrypto*, *M2crypto*, *construct* and *progressbar*).

```
> sudo ARCHFLAGS='-arch i386 -arch x86_64' easy_install pycrypto
> sudo easy_install M2crypto construct progressbar
```

Download and install Mercurial (http://mercurial.selenic.com/) to check out the source code from the iphone-dataprotection Google code repository.

```
> hg clone https://code.google.com/p/iphone-dataprotection/
> cd iphone-dataprotection
```

Decrypting Normal backups:

In case of normal backups, the data protection class keys stored in the Backup Keybag are protected by a hardware generated key (Key 0x835). In order to grab the protection class keys from the Backup Keybag Key 0x835 is required and the key is computed only on the device. So decryption of protected files in the normal backup is not possible without having access to the actual device. In forensic investigations the information recovered from the normal backups is less if physical access to the device is not available.

Steps below explain the procedure to decrypt the protected files stored in the normal backup in case physical access to the device is obtained. On the iPhone, Key 0x835 is computed by the IOAESAccelerator kernel service at iOS boot by encrypting a static value 01010101010101010101010101010101 with UID. UID is a hardware encryption key embedded in the iPhone application processor AES engine and it is unique for each device. iOS running on the iPhone cannot read the hardware key (UID) but it uses the key to compute Key 0x835 in kernel mode. UID is not accessible to user land process. This restriction can be bypassed by patching the IOAESAccelerator kernel service.

Steps to extract Key 0x835 from the iPhone:

1. Jailbreak your iPhone.

**If you don't like to Jailbreak the phone, follow the steps explained in the iPhone Forensics article and grab the encryption keys.*

2. On the iPhone, install OpenSSH from Cydia. OpenSSH allows connecting to the device over SSH.
3. On Mac OS X workstation, download device_infos, kernel_patcher and Cyberduck tools.

4. Connect the iPhone and workstation to the same Wi-Fi network.
5. On OS X run Cyberduck and connect to the iPhone by typing iPhone IP address, *root* as username and *alpine* as password.
6. Copy *device_infos* and *kernel_patcher* executables to the iPhone root directory.
7. Run Mac terminal and SSH to the iPhone by typing iPhone IP, *root* as username and *alpine* as password.

```
> ssh  root@iPhone-IP
  Password: alpine
```

8. On SSH terminal, run the below commands to change the execution permissions of *kernel_patcher* and *device_infos*.

```
> chmod 777 kernel_patcher
> chmod 777 device_infos
```

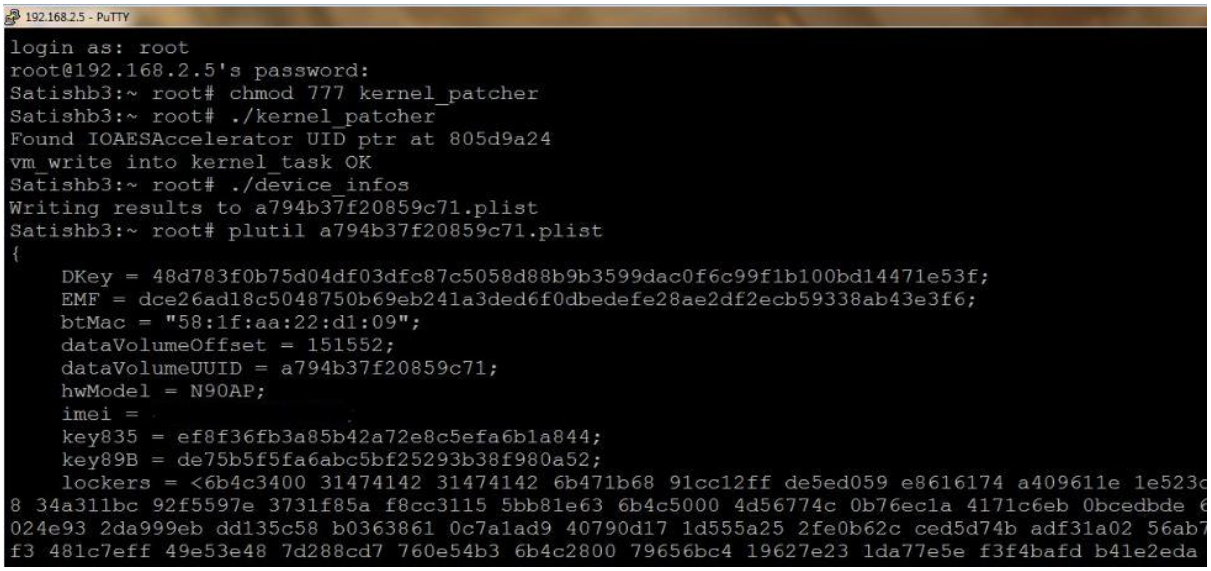
9. Patch IOAESAccelerator kernel service to use the hardware encryption key (UID) from user land process. *Kernel_patcher* script modifies the kernel and applies the required patches to IOAESAccelerator.

```
> ./kernel_patcher
```

* If the kernel is already patched, the above script displays *kernel patching failed* message.
** Kernel_patcher script only works for iOS 5 devices*

10. Run *device_infos* script and supply *key835* as a parameter. The script computes the Key 0x835 and displays on the screen. If *key835* parameter is not supplied, the script computes all the encryption keys and stores them in a Plist file (Figure 9).

```
> ./device_infos key835
```



(Figure 9)

Once Key 0x835 is grabbed, it is possible to decrypt the Backup Keybag and obtain the data protection class keys. Later these class keys are used to decrypt the protected files in the backup.

11. On Mac OS X terminal, navigate to iphone-dataprotection directory. Run the *backup_tool.py* script by supplying the iTunes backup directory path.

```
> python python_scripts/backup_tool.py /Users/User/Library/Application
      Support/MobileSync/Backup/[iPhone UDID]/ [output_path]
```

** If output_path is not mentioned, the script creates [iPhone UDID]_extract directory in the backup folder and extracts the backup files into it.*

On the backup, the iPhone keychain sqlite database is stored as a Plist file (Keychain-backup.plist). The Plist file contents are encrypted with the keychain data protection class keys. Items in the keychain can only be viewed after decrypting it with the keychain protection class keys.

Run *keychain_tool.py* and supply *Key 0x835*. The script decrypts the Backup Keybag, grabs the protection class keys from 6 to 11 (listed in Table 6) and decrypts the keychain items.

```
> python python_scripts/keychain_tool.py –d /Users/User/Library/Application
      Support/MobileSync/Backup/[iPhone UDID_extract]/keychain-backup.plist
      /Users/User/Library/Application
      Support/MobileSync/Backup/[iPhone UDID_extract]/Manifest.plist
```

The above script dumps the generic passwords, internet passwords, certificates and private keys from the keychain backup file.

Decrypting Encrypted backups:

In case of encrypted backups, migratable data protection class keys (6 to 8 listed in Table 6) stored in the Backup Keybag are protected by iTunes password and ThisDeviceOnly class keys (9 to 11 listed in Table 6) stored in the Backup Keybag are protected by Key 0x835 along with the iTunes password. Most of the data stored in the encrypted backups is migratable as the data is encrypted with the iTunes password and it is not tied to a specific device. Files in the backup are encrypted with a unique key for each file using AES 256 in CBC mode. Encryption keys are stored in the Backup Keybag and protected by iTunes password. In order to decrypt the Backup Keybag, grab the protection class keys and decrypt backup files iTunes password is required. So decryption of files in the encrypted backup is not possible without the iTunes password. In forensic investigations the information recovered from the backups is less if the iTunes password is not available. As iTunes does not impose any password strength rules on encrypted backups, it is easy to perform a brute force attack and predict the password. Encrypted backups add a significant difficulty in data recovering and it may be impossible with a complex password in use.

During the backup iTunes stores the encrypted backup password on the iPhone keychain. So if the backup password is unknown and physical access to the device is available, the backup password can be retrieved by viewing the iPhone keychain items. On a JailBroken iPhone, all the keychain items can be viewed using keychain_dumper tool. Usage of this tool is documented at keychain-dumper-usage post.

Tools like iPhone Backup Extractor & iPhone Backup Browser does not work on encrypted backups. They can only read & parse the Manifest.mbdb file and prepares a file structure. However the file cannot be opened as the content is encrypted.

Steps below explain the procedure to decrypt the files stored in the encrypted backup with a known iTunes password.

Run *backup_tool.py* and supply iTunes password to it. In case if the password is unknown, modify the backup_tool.py script and attach a brute force script to it. *Backup_tool.py* script takes the user entered password, decrypts the Backup Keybag, grabs all the encryption keys and decrypts the files in the backup.

```
> python python_scripts/backup_tool.py /Users/User/Library/Application
Support/MobileSync/Backup/[iPhone UDID]/ [output_path]
```

** If output_path is not mentioned, the script creates [iPhone UDID]_extract directory in the backup folder and extracts the backup files into it.*

On the encrypted backup, the iPhone keychain sqlite database is stored as a Plist file (Keychain-backup.plist). The Plist file contents are encrypted with the migratable and ThisDeviceOnly keychain data protection class keys.

To view migratable keychain items run *keychain_tool.py* and supply iTunes password.
To view ThisDeviceOnly keychain items run *keychain_tool.py* and supply *Key 0x835.*

```
> python python_scripts/keychain_tool.py –d /Users/User/Library/Application
      Support/MobileSync/Backup/[iPhone UDID_extract]/keychain-backup.plist
      /Users/User/Library/Application
      Support/MobileSync/Backup/[iPhone UDID_extract]/Manifest.plist
```

The above script dumps the generic passwords, internet passwords, certificates and private keys from the keychain backup file.

Forensic investigation on the backup files would allow examiners to gain access to the entire contents of its host phone until the point that the backup took place. It is also quite possible that the seized system might contain older copies of the backup files or other iPhone backups with a wealth of information.

To view the list of available backups on a system, open iTunes and navigate to *Edit->Preferences* (on windows) *or iTunes->Preferences* (on Mac) menu and choose *Devices* tab. It displays the list of backups as shown in the Figure 10.

(Figure 10)

iTunes also provides an option to delete the backup files. To delete an existing iPhone backup, in the Devices Preferences window (shown in the above screenshot) select a backup and click on *Delete Backup...* button. If a backup is deleted from a system, examiners can use data recovery or carving

tools to recover the deleted files from the system hard disk. It is easy to recover the deleted files from the computer when compared with iPhone.

The iPhone stores a lot of user data in the backup files. The following table list out the common sources of potential evidence that can be analyzed in an investigation.

File Name	Description
AddressBook.sqlitedb	Contact information and personal data like name, email address, birthday, organization, etc...
AddressBookImages.sqlitedb	Images associated with saved contacts
Calendar.sqlitedb	Calendar details and events information
Call_history.db	Incoming and outgoing call logs including phone numbers and time stamps
Sms.db	Text and multimedia messages along with their timestamps
Voicemail.db	Voicemail messages
Sfari/Bookmarks.db	Saved URL addresses
Safari/History.plist	User's internet browsing history
Notes.sqlite	Apple Notes application data
Maps/History.plist	It keeps track of location searches
Maps/Bookmarks.plist	Saved location searches
consolidated.db	Stores GPS tracking data
En_GB-dynamic-text.dat	Keyboard cache
com.apple.accountsettings.plist	Maintains data about all email accounts that are configured on the Apple Email application
com.apple.network.identification.plist	Wireless network data including IP address, router IP address, SSID and timestamps

(Table 7)

Along with the files listed in the above table, the iPhone backup also contains third party application files. Sensitive information stored in the third party application files may also provide possible evidence for the investigation.

Example: Facebook and LinkedIn iPhone applications store the authentication tokens and cookie values in plist files on the device. During backup, iTunes copies the plist files on the device to the backup folder. In such cases, analyzing the backup files gives access to the authentication tokens which in turn allows to log into the application without supplying the username and password. More details about Facebook plist hijacking are documented at scoopz blog.

During an iPhone backup, iTunes only stores the existing files to the backup. So it is not possible to recover the deleted files on the iPhone from backups. Though it is not possible to recover the deleted iPhone data, forensic examiners prefer analyzing the backups to collect the evidence as it does not compromise the contents on a live device.

Video:

Video - http://www.youtube.com/watch?v=cqj4Z8VkJyU&feature=plcp
Techniques explained in the article are demonstrated in the video.

Video Transcript - *http://securitylearn.files.wordpress.com/2012/06/analysis-of-ios-backups-video-transcript.docx*

Conclusion:

Techniques illustrated in the article shows that forensics investigation is possible on the latest version of iPhone backups. However the information recovered from the backup alone without physical access to the device is less. Apple is also changing the backup mechanism with every major release of iTunes. So it is always challenging to design the scripts to decrypt the iTunes backups.

References

1. iPhone data protection in depth by Jean-Baptiste Bédrune, Jean Sigwald
   http://esec-lab.sogeti.com/dotclear/public/publications/11-hitbamsterdam-iphonedataprotection.pdf
2. iPhone data protection tools
   http://code.google.com/p/iphone-dataprotection/
3. iPhone wiki
   http://theiphonewiki.com
4. Processing iPhone backups
   http://www.appleexaminer.com/iPhoneiPad/iPhoneBackup/iPhoneBackup.html

About Me:

Satish B (@satishb3) is an Information Security Professional with 6 years of experience in Penetration testing of web applications and mobile applications.

My blog is located at http://securitylearn.wordpress.com
Email: satishb3@hotmail.com