Mobile Device Encryption Systems

Peter Teufl, Thomas Zefferer, Christof Stromberger

▶ **To cite this version:**

Peter Teufl, Thomas Zefferer, Christof Stromberger. Mobile Device Encryption Systems. 28th Security and Privacy Protection in Information Processing Systems (SEC), Jul 2013, Auckland, New Zealand. pp.203-216, 10.1007/978-3-642-39218-4_16 . hal-01463828

HAL Id: hal-01463828
https://inria.hal.science/hal-01463828

Submitted on 9 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Mobile Device Encryption Systems

Peter Teufl, Thomas Zefferer and Christof Stromberger
{peter.teufl, thomas.zefferer}@iaik.tugraz.at, stromberger@student.tugraz.at

IAIK, Graz University of Technology
Inffeldgasse 16a, 8010 Graz, Austria

Abstract. The initially consumer oriented iOS and Android platforms, and the newly available Windows Phone 8 platform start to play an important role within business related areas. Within the business context, the devices are typically deployed via mobile device management (MDM) solutions, or within the bring-your-own-device (BYOD) context. In both scenarios, the security depends on many platform security functions, such as permission systems, management capabilities, screen locks, low-level malware protection systems, and access and data protection systems. Especially, the latter play a crucial rule for the security of stored data. While the access protection part is related to the typically used passcodes that protect the smartphone from unauthorized tempering, the data protection facility is used to encrypt the core assets – the application data and credentials. The applied encryption protects the data when access to the smartphone is gained either through theft or malicious software. While all of the current platforms support these systems and market these features extensively within the business context, there are huge differences in the implemented systems that need to be considered for deployment scenarios that require high security levels. Even under the assumption, that the underlying encryption systems are implemented correctly, the heterogeneity of the systems allows for a wide range of attacks that exploit various issues related to deployment, development and configuration of the different systems.

In order to address this situation, this paper presents an analysis of the access and data protection systems of the currently most popular platforms. Due to the important influence of the developer on the security of the iOS Data Protection system, we also present a tool that supports administrators in evaluating the right choice of data protection classes in arbitrary iOS applications.

1 Introduction

The recent success story of smartphones and tablets, which will be referred to as mobile devices for the remainder of this document[1], was mainly fueled by user-friendly and consumer-oriented devices introduced by Apple and Google in 2007

[1] Due to restricting the analysis to iOS and Android devices, a distinction between smartphones and tablets is not required.

and 2008. For the past years, Apple's iOS platform and Google's Android platform have been dominating the market. During this time, both companies have introduced a wide range of security related features for their smartphone platforms in order to make their platforms ready for business applications. Recently, the market power of iOS and Android has been challenged by a new Windows Phone 8 release[2] and a new version of RIM's BlackBerry platform. For all major smartphone platforms, encryption represents a core feature that is advertised for its strong security[3][4][5]. However, encryption systems of smartphone platforms differ in various security related aspects. For instance, different platforms rely on different approaches to encrypt data (file based encryption vs. file-system based encryption) and implement different methods to derive required encryption keys from user input (e.g. PIN or passcodes). Furthermore, different platforms offer both developers and end users different options to use and configure provided encryption features. The choice of these parameters also significantly influences the security of provided encryption systems.

As a consequence of these differences, a direct comparison of different platforms is difficult. However, understanding capabilities and limitations of encryption systems provided by smartphone platforms is crucial when deploying these platforms in security-critical areas and allowing mobile devices to access confidential data. While general security assessments of smartphone platforms have already been presented in literature [4] [11] [9], no in-depth assessment and comparison of different encryption-systems is available so far. Such an assessment could serve as a basis of decision-making and help administrators in charge to choose the platform that best meets the given requirements.

To bridge this gap, we propose an abstract assessment model for encryption systems provided by smartphone platforms. Based on experience gained during the development of a security-critical application[6], the proposed assessment model defines generic properties of encryption systems provided by nowadays smartphone platforms. The proposed abstract model can be applied to arbitrary smartphone platforms in order to assess and compare capabilities of their encryption systems. In this paper, we use the proposed model to assess and compare the encryption systems of Apple iOS and Google Android. These platforms have been chosen due to their predominating market share and their growing importance for business applications. The Windows Phone 8 and BlackBerry platforms have not been considered so far, since detailed information on the platforms' encryption systems has not been made publicly available so far. As soon as this information is published, the proposed assessment model can be ap-

[2] Due to the lack of its business related features, such as mobile-device-management and encryption, Windows Phone 7 is not considered here.

[3] http://www.apple.com/ipad/business/ios/

[4] http://www.samsung.com/global/business/mobile/product/smartphone/GT-I9300MBDXSP-features

[5] http://www.windowsphone.com/en-us/business/for-business

[6] SecureSend for iOS https://itunes.apple.com/us/app/secure-send/id560086616?mt=8, Android and Windows Phone versions are currently under development.

plied to these platforms as well. For the time being, this paper focuses on Apple iOS and Google Android and provides an detailed assessment of these platforms' encryption systems.

2 Related Work

Due to their growing relevance for security-critical applications, the security of Google Android and Apple iOS is also of increasing interest for the scientific community. A general comparison between Google Android and Apple iOS is for instance given by Goadrich et al. in [5]. However, their work focuses rather on differences in application development between these two platforms than on security issues. Security aspects of application development under Android have been discussed in more detail by Enck et al. in [4]. More general assessments of Android's security features have been provided by Shabtai et al. [11] and by Pacatilu in [8]. Similar to the Android platform, also the security of the iOS platform has been discussed in literature. A comprehensive analysis of possible attacks on the iOS platform has for instance been provided by Pandya in [9].

Most of the above mentioned publications have discussed, analyzed, and assessed the security of Google Android and Apple iOS on a rather general level. For our contribution, related work dealing with encryption systems for mobile devices is of special interest. Indeed, various authors have approached this topic from different perspectives so far. The relevance of encryption solutions on mobile devices and possible implications on jurisdiction have been discussed by Paul et al. in [10]. Proprietary encryption solutions for smartphone platforms have for instance been proposed in [12] and [3].

Interestingly, most related work on encryption systems on mobile devices focuses on the development of proprietary solutions. From a application developer's point of view, it is however more convenient to rely on encryption functionality provided by the underlying smartphone platform instead of implementing own encryption systems. Furthermore, implementing own encryption solutions carries the risk of making implementation errors that again can compromise security. Due to these reasons, relying on integrated encryption systems provided by the underlying smartphone platform can be advantageous in most cases. However, reliance on provided encryption systems requires detailed knowledge of their capabilities and limitations. We propose an abstract and platform-agnostic assessment model for the evaluation of encryption systems provided by smartphone platforms in the next section.

3 Assessment Model

Several architectural and conceptual differences between encryption systems of different smartphone platforms render systematic assessments and direct comparisons difficult. In order to allow for systematic assessments, we therefore extract common properties of nowadays systems and combine them with generic security considerations. This way, we first derive a platform-agnostic encryption

model and identify common assets and threats of encryption systems for mobile devices. From the identified threats, generic attack scenarios are then derived. The proposed abstract encryption model and the derived generic attack scenarios can finally be used to systematically assess and compare specific encryption systems of arbitrary smartphone platforms.

Figure 1 shows an abstract and platform-agnostic model that covers typical capabilities of nowadays smartphone platforms' encryption systems.

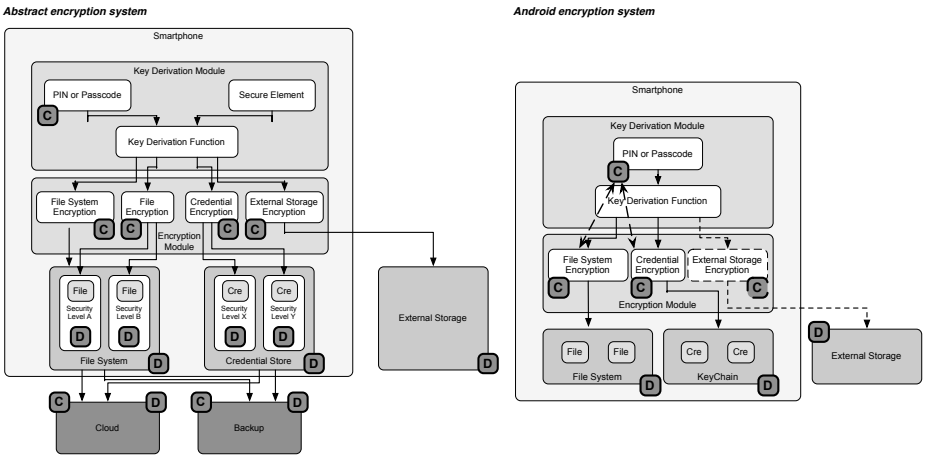

Fig. 1: Abstract encryption model (left), Android encryption system (right).

As shown in Figure 1, mobile devices typically provide three different locations to store data. Data can either be stored in the local *file system*, in a dedicated *credential store* provided by the platform, or in an *external storage* such as a microSD card. To encrypt data being stored at these locations, smartphone platforms feature some kind of *encryption module*. Depending on the smartphone platform, the encryption module contains one or more submodules to completely encrypt an entire storage location, or to encrypt specific files and credentials residing at a certain storage location. The different encryption modules need to be supplied with appropriate encryption keys. These keys are provided by the *key derivation module*. This module implements a key derivation function that derives required encryption keys from different inputs. Potential inputs are PINs or passcodes defined and entered by the user, or master keys stored in secure elements. The platform-agnostic model shown in Figure 1 also includes further external components such as *backup* and *cloud* components. These external components also need to be considered when assessing encryption systems, as data is potentially transferred to these external entities.

However, the security of data and/or credentials stored on mobile devices does not solely depend on the components of the smartphone's encryption system. Additionally, configuration options defined by administrators or users, and decisions made by application developers on how to use functionality provided

by the given encryption system can also influence the capabilities of encryption systems and hence the security and confidentiality of data stored on mobile devices.

Components of the encryption system that are subject to configuration options (C) and developer decisions (D) are marked accordingly in Figure 1. The user and/or administrator influences the strength of the used PIN or passcode, and enables or disables file system encryption and encryption of external storages manually or via mobile device management (MDM) solutions. Also, application developers need to decide where to store data (file system, credential store, external storage). Additionally, developers can choose to rely on a file based encryption of data and are responsible to select appropriate security levels for encrypted files. Finally, developers can also decide whether data is transferred to external cloud or backup components.

3.1 Assets and Threats

From the platform-agnostic encryption model shown in Figure 1, general assets and threats can be derived. In general, encryption systems are developed and used to assure the confidentiality of data. Hence, data represents the primary assets that needs to be considered for systematic assessments of encryption systems on smartphone platforms. The security of encryption systems on smartphone platforms and the confidentiality of the asset data can be compromised by different threats. Due to their mobility and their broad support for third-party applications, *theft* and *malware* represent the main threats for mobile devices. Hence, encryption systems for mobile devices need to be designed such that encrypted data stored on the smartphone cannot be decrypted by an illegitimate user or by malware running on the mobile device.

3.2 Assumptions

Based on the two basic threats *theft* and *malware*, we define assumptions that define the scope of the conducted security assessment. In particular, the conducted assessments are based on the following three assumptions.

First, our assessments are based on the assumption that all cryptographic algorithms used by the assessed encryption systems are correctly implemented by the smartphone platform. The goal of this assessment is not to evaluate the correct implementation of particular algorithms but to analyze weaknesses of the respective encryption systems that can be exploited by an attacker to gain access to the asset data. **Second**, we assume that an attacker steals a smartphone with the intention to gain access to data stored on the device. We further assume that the attacker is an expert who knows the deployed encryption system and its weaknesses, and thus is capable of either circumventing the encryption system or mounting brute-force attacks on the passcode. In case of theft, we assume that the smartphone is locked via a PIN or passcode. **Finally**, malware is only considered in this paper within the scope of jailbreaking a locked (or switched off) stolen mobile device. Other types of malware that are used to attack a user

directly without stealing the mobile device is not considered for the following reasons: Malicious software that uses root exploits to gain access the the operating system is not limited in its capabilities and can attack the data directly on the smartphone. Another category of malicious software only relies on provided system APIs without exploiting a security vulnerability to gather pre-attack information about the passcode (e.g. via phishing, or retrieving information about the passcode complexity). This information can then be used when the device is stolen by an attacker. However, this type of malware is also not considered in this work, since a detailed description of such attacks would be beyond the scope of this work.

3.3 Attack Scenarios

Based on the defined assets, threats, and assumptions, a set of generic attack scenarios can be derived from the platform-agnostic encryption model shown in Figure 1.

Attacks on the encryption system include attacks on properties of the encryption system and its integration into the platform. The following specific attacks need to be considered here: (1) Circumventing the encryption system by utilizing *jailbreaking/rooting* on a stolen smartphone, (2) *attacking backups* that are either stored on disk or in the cloud, and (3) *attacking cloud storage* that is provided by the platform for data-synchronization purposes.

Under the assumptions that the encryption system is implemented correctly, **attacks on key derivation** are considered to be the most likely attacks: (1) Some encryption systems do not use the user's passcode to derive encryption keys, which enables *jailbreaking/rooting* attacks, (2) even when the passcode is used for deriving encryption keys, the system is still susceptible to *brute-force attacks* on the passcode. The time required to carry out such attacks primarily depends on the employed key derivation function, and the inclusion of a secure element in the key derivation process.

Finally, **attacks on user configurations or developer decisions** need to be considered due to the various properties and parameters that can be influenced by the administrators, the users and the developers. Depending on the specific properties of the system (e.g. brute-force times on the passcode), appropriate passcodes must be chosen, or the system might not be enabled by default (*poor configuration option*). Application developers can influence the way, in which smartphone applications make use of available security features. Depending on the particular platform, an application developer can decide where to store data, which security level to use, and whether data is transmitted to external backup and cloud components. If *poor developer decisions* are made, attacks can potentially circumvent integrated security features.

In this section, we map the abstract encryption model and the generic attack scenarios defined above to the Google Android platform (version 4.2). Figure 1 shows the result of this mapping process and illustrates relevant components of Android's encryption system. In contrast to the iOS system, the Android backup system (Google Backup) is not considered in this analysis due to two reasons: It

is not activated per default and local backups, which play an important role for stolen devices, are not available in the raw Android version by Google.

3.4 Encryption System

Android uses a file-system based encryption system based on the dm-crypt transparent disk-encryption system[7] that has been available in the Linux kernel since version 2.6. By using the Linux kernels's device-mapper functionality, the encryption layer can be added between the file-system and the actual block-device that stores the raw data.

Android derives the keys for the file-encryption system from the PIN/passcode of the user during system startup. In contrast to other platforms such as iOS, no secure element is involved in this key-derivation procedure. For more detailed information on the Android encryption system and design considerations the reader is referred to the Android documentation[8].

In addition to this system, an additional system for the secure storage of private cryptographic keys is provided by Android. This secure storage is called Android KeyChain and is publicly accessible to third-party applications since Android 4.0. Keys stored in the Android KeyChain are encrypted with AES. The encryption key is derived from the user's PIN or passcode that is used to unlock the smartphone.

In general, external storage on Android devices (e.g. microSD cards) are not protected by the Android file-encryption system. Furthermore, there is no access control on these areas beyond the permissions, which are required to read or write to this storage. Some smartphone vendors have extended the functionality of Android to support encryption of external storage media. However, a detailed analysis of these vendor-specific approaches is beyond the scope of this paper.

As all used encryption keys for file-system encryption and KeyChain are derived from the user's PIN/passcode, these encryption systems are not vulnerable to **jailbreaking/rooting**. Even if an attacker gains root access to the smartphone, the user's PIN/passcode that is required to derive encryption keys remains unknown.

3.5 Key Derivation

As briefly mentioned above, the key derivation on Android is based on the PIN/passcode of the user. For the encryption and decryption process, the password of the user is combined with a salt value that is stored in the encryption footer of the file system. The resulting value is then used as input for the PBKDF2 function, which basically applies SHA1 repeatedly. The result represents a 128 bit AES key, that is used to decrypt the 128 bit AES master key for file-system encryption and for the protection of KeyChain entries.

[7] http://code.google.com/p/cryptsetup/wiki/DMCrypt
[8] http://source.android.com/tech/encryption/android_crypto_implementation.html

Since Android does not include a secure element, the user's PIN/passcode is the only unknown in the key-derivation process. Thus, the key-derivation process is not bound to the mobile device and can also be out-sourced to external more powerful entities. This significantly facilitates the accomplishment of **brute-force attacks** on the user's PIN/passcode and potentially decreases the security of Android's key-derivation method.

3.6 Configuration Options and Developer Decisions

The security of Android's encryption system heavily depends on the configuration chosen by the user or a mobile device management (MDM) system. On Android, the file-system encryption is not enabled by default and must either be activated by the user or by the respective policy of the MDM system. Before the encryption system is activated, a password needs to be defined by the user. The security of the whole system primarily depends on the length and quality of this password. In general, the user has the choice to enter a simple numerical PIN code or a more secure alpha-numeric passcode. In managed envrionments, a minimum quality of the user's PIN or passcode can be enforced by appropriate MDM policies.

Beside the user, also application developers can influence the security and confidentiality of data being stored on smartphones. First of all, a developer needs to decide where to store confidential data. Depending on the chosen storage location, different encryption schemes are applied to the stored data. As discussed above, the availability of encryption systems on Android also depends on user configurations. Hence, application developers must not assume that certain encryption features such as file-system encryption are enabled on the target device.

4 iOS Analysis

The analysis of the iOS system is more complex than the Android analysis, because there are three systems (Figure 2) that need to be considered for data and credentials protection (two, when external backups are not counted). Especially, the file-based data protection system offers a high level of security due to the inclusion of a secure element. However, this high level of security can only be achieved when the right configuration and developer choices are made. The information in the subsequent analysis is based on Apple documentation [1], third-party analysis [6], [13], and our own analysis within the context of secure application development and external consulting projects.

4.1 Encryption System

The first **file-system encryption system** – depicted in the left part of Figure 2 – is available since the iPhone 3gs (iOS 3.x) and encrypts the whole file-system. The file system key (EMF key) is randomly created when the device is started for the very first time. It is stored in the so-called *Effaceable Storage*, which is a

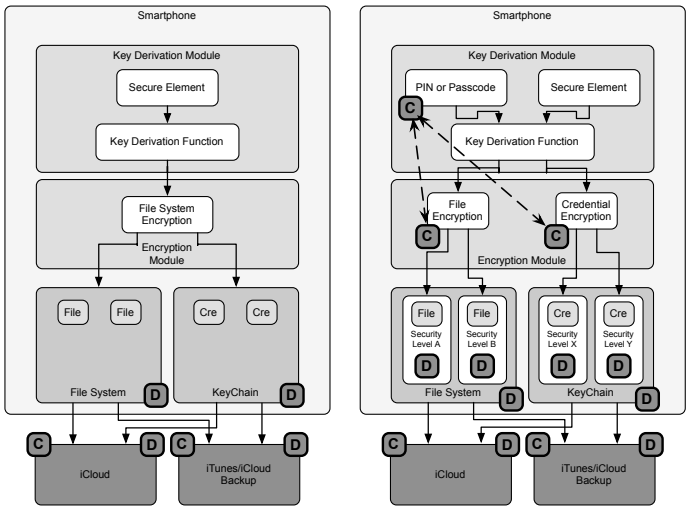

Fig. 2: iOS encryption systems: The always-on file-system based system is shown in the left part, the file-based data protection system is depicted on the right.

part in the flash memory that can be wiped very fast. This capability is employed for fast remote wiping, which only deletes the cryptographic keys instead of the whole file-system. The EMF key itself is encrypted by the unique device identifier (UID) AES key, which is stored within a secure element. The employment of a secure element eliminates the possibility to gain access to file-system images that are either gained by cloning or ripping out the flash memory. In other words, any attack must be executed on an iOS device due to the presence of the secure element.

The system can easily be attacked via **Jailbreaking/Rooting**, which concentrates on the weakest point of the file-system based encryption system: The protection of the EMF key for encrypting the file-system is not based on the passcode of the user, but relies only on the integrated secure element. Thus, even when a passcode is set on the device, the application of a jailbreak enables the attacker to gain root access to the operating system, which decrypts the data with the EMF key. Thus, an attacker can gain access to all data without knowing the encryption keys. Due to the availability of jailbreaks for almost all iOS versions[9], the execution of such an attack does neither require in-depth knowledge nor sophisticated resources.

The second system – the **Data Protection API** – has been introduced with iOS 4 for protecting individual files and credentials. The system is based on various protection classes that need to be defined by the developer for stored files and credentials. These protection classes define when the respective file-system encryption keys are available and can be used for decrypting the protected

[9] A good overview is given at: http://www.apfelzone.at/jailbreak-ubersicht/

files. The four available data (file) protection classes are *NSFileProtection*{*None, Complete, UntilFirstUserAuthentication, CompleteUnlessOpen*}. Thereby, *None* indicates that the file is not specifically protected and the only protection is offered by the file-system based encryption system. *Complete* means that the file-encryption keys are removed from memory whenever the device is locked. *UntilFirstUserAuthentication* decrypts the file-system encryption keys when the passcode is entered the first time after a device reboot. The decrypted keys are then kept in device memory until the next shutdown. Finally, *CompleteUnlessOpen* is used to write data to open files when the device is locked and in addition offers a system based on asymmetric cryptography that is used for encrypting data which is received while the device is locked (e.g. emails).

iOS offers the KeyChain which encrypts and stores credentials, such as private keys, passwords, certificates etc. There are similar protection classes for protecting these credentials: *kSecAttrAccessible*{*Always, WhenUnlocked, AfterFirstUnlock*}. Thereby, their functionality corresponds to the first three data protection classes. An important difference is that the KeyChain protection classes also exist in a *DeviceOnly* version which indicates that the credentials cannot be transferred off-device via iCloud or iTunes backups. A detailed discussion of these protection classes the reader is referred to [2]. The most important aspect is that for the protection classes other than *None, Always* the keys required for decryption are derived by utilizing the secure element *and* the user's passcode.

Considering **Jailbreaking/Rooting** the following conclusions can be drawn: When the *None, Always* protection classes are employed, the protection is equal to that offered by the file-system based encryption system and can easily be circumvented by applying a jailbreak to the system. The problem is extensively discussed in [6].

4.2 Key Derivation

Strictly speaking the term "key derivation" is not adequate for the description of the **file-system encryption system**. The reason is, that the UID key within the secure element is used to encrypt/decrypt the actual file-encryption master key (EMF key). Thus, there is no typical key-derivation function involved. Since, the previously described jailbreaking/rooting attack can easily be deployed to gain access to the file-system, a specific attack on this "key derivation" system is not necessary, and thus not further considered.

The **Data Protection system** employs the standardized *Password Based Key Derivation Function 2* (PBKDF2)[1], which is specified in PKCS#5[7] as key derivation function. The user's passcode is tangled with the UID key stored in the secure element and combined with a salt. The resulting value is then used as input for the key derivation function with an iteration count of 10,000. The gained key is used to encrypt and decrypt the aforementioned protection class keys.

When applying a jailbreak to the iOS device, which is protected by the Data Protection system, the attacker has access to the file-system. However, the files and credentials that use the correct protection classes (other than *None*

for files, *Always* for KeyChain entries) cannot be decrypted without knowing the passcode. Under the assumption, that this passcode is not known to the attacker the only remaining option for the attacker is the application of a **brute-force attack**. Due to usability issues with long and complex passcodes, the attacker can assume that only rather short ones are utilized. Although, this reduces the number of possible passcodes, the presence of the secure element and the PBKDF2 key derivation function significantly slows down the brute-force attack. Due to the high iteration count of PBKDF2, the derivation of an AES key from the passcode takes roughly 90 ms[10]. This delay can be used as a basis for calculating the worst-case brute-force attack times when choosing a password. Also, and probably even more important, the brute-force attack must be carried out on the device, since the secure element is also involved in the key derivation process. Thus, the brute-force attack cannot be sped up by using external processing resources and must be carried out on the device. The described attack is implemented by a forensic toolkit offered by the UK based company Elcomsoft[11].

Although the **backup encryption system** is technically not a part of the mobile device, it still plays an important role for the security of the data and the credentials. On iOS, the backup can either be made via an iTunes installation, or to the Apple based iCloud solution. Thereby, the iTunes backup can either be stored in plain text or be encrypted with a key derived from a user based password. This key derivation is also based on the PBKDF2 function. However, and this is the most important difference, due to the lack of a secure element on the PC/laptop where iTunes is installed, the key derivation subsystem is less protected than its counterpart on the iOS device.

The backup on an iTunes device can be attacked via different techniques. (1) in case of unencrypted backups the attacker can just copy the whole backup and get access to all of the files that are marked for backup by the installed iOS applications (see below). (2) in case of an encrypted backup, the attacker can carry out a **brute-force attack** on the encryption password. Due to the lack of a secure element, the attacker can use external processing resources to speed up the brute-force attack. This scenario is basically identical to the brute-force attack on the Android encryption system.

4.3 Configuration/Developer

The *file-system* based encryption system of iOS cannot be configured, since it is enabled per-default and cannot be deactivated. The *data protection* system is activated as soon as device passcode is set, *and* if the developer has chosen the correct protection classes for the application files. Since, the required protection classes can neither be configured by the user nor the administrator (MDM), the most important configuration options are the password properties, such as length, complexity class and the automated screen lock functionality and the

[10] http://www.securitylearn.net/tag/iphone-data-recovery-on-ios-5/
[11] http://www.elcomsoft.co.uk/eift.html

related timeout values[12]. The user/administrator is capable of configuring the options for the third system – the *backup functionality*. Here, the configuration options are related to storing encrypted or plain backups on iTunes, and activating/deactivating the backup functionality for iTunes and iCloud respectively. Especially the encrypted iTunes backup and the iCloud backup depend on the properties of the chosen passcodes. Thereby, the iTunes passcode is chosen by the user on the laptop/PC where the backup is stored. For iCloud backups the security of the password for the required iCloud account is considered as vital.

Poor configuration options are especially critical in the unmanaged scenario where the user selects the passcode, the attacker can base an attack on the data protection system and the backups on the following assumptions: (1) Due to usability issues related to passcode length and complexity class the user will typically not use a passcode or select a rather short one. (2) Since the default setting for PIN locks on an iOS device uses 4-character numerical PIN codes, which can easily be verified by the attacker when looking at the type of used lock screen, a brute-force attack might be feasible. (3) For the iTunes related backup the administrator/user choice regarding the activation/deactivation of the backup, and the user's choice of the backup encryption password play an important role within security. (4) The user/administrator also decides whether the iCloud-based system is allowed. In case of its availability a weak password of the associated iCloud account, can used as an attack path by an attacker do gain access to the backups.

On iOS, the developer influences some vital aspects of the encryption system security. (1) The developer specifies the protection classes for specific files or KeyChain entries. When the wrong classes (especially *None* and *Always*) are chosen by the developer the security is reduced to that of the file-system based encryption, which can easily be circumvented. (2) The developer also decides whether a file is included in iTunes/iCloud backups. Critical files that should not be included in backups, must be explicitly marked by the flag *kCFURLIsExcludedFromBackupKey*. Unfortunately, neither the protection classes of application files nor the backup flag can directly be inspected by the user/administrator. This opens a critical security issue, that could easily be exploited by an attacker. Assuming an application (e.g. the Apple mail) application uses the right protection classes, further assuming a user opens an attachment (e.g. a PDF file) in an external application that employs the *None* protection class for storing the file, then an attacker does not need to apply a brute-force attack on the passcode in order to get access to the email with the PDF document. Instead, the attacker can simply apply a jailbreak and extract the file form the external application that was used to view/edit the PDF file.

In order to mitigate of the threat, we have created a simple Java based tool[13] for determining the protection classes of the installed applications. The tool ex-

[12] The available setting can be seen in the Apple Configuration Application, if no MDM-system is used. https://itunes.apple.com/en/app/apple-configurator/id434433123?mt=12

[13] https://github.com/ciso/ios-dataprotection/

tracts the protection classes of each file on the iOS system from an existing iTunes backup, and thus allows the user/administrator evaluate the security of the installed applications. Obviously, the protection classes can only be determined for those files that are included in an iTunes backup.

Attack	Brute-force attacks	Jailbreak/Rooting	Poor developer's choice	Poor configuration options
Applies to	Key Derivation	Encryption System	Developer	Configuration
Android				
File System	off-device	brute-force required	no influence	default: off
KeyChain	off-device	brute-force required	developer	passcode is mandatory
External storage	not encrypted	not encrypted	developer	not consistent
iOS				
File system	not required	direct access to data	always on	no influence
Data protection (files, KeyChain)	on-device	brute-force required	developer	only on when passcode is set
External storage	NA	NA	NA	NA

Fig. 3: Comparision between iOS and Android encryption systems. The black, light-grey and white cells indicate critical, medium, and minor issues. The dark-grey cells indicate that the attack is not relevant for the given property or the analyzed platform.

5 Conclusions

The conducted analysis shows, that although encryption systems are present on all current platforms, their heterogeneity causes security issues that need to be considered when deploying a mobile device platform. When looking at iOS and Android, the following summary can be given. Due to the strong key derivation function based on the user's passcode and the device's secure element, the iOS systems offer a good level of protection. However, this level can only be achieved when the developer as well as the user/administrator make the right decisions. Since, there are multiple systems that need to be considered, an in-depth knowledge is required by the developer and the user/administrator. One of the most disturbing facts is that neither the user nor the administrator can verify whether an application uses the appropriate protection classes. We have addressed this problem by creating a backup analysis tool that extracts the protection classes of application files, which can then be used to asses the security of application data. On Android, the employed encryption system is much simpler than that on iOS. The main advantage is that the the file-system based encryption system requires a passcode during boot-up which offers an adequate protection against jailbreak attacks that require a system reboot. On the other hand, due to the lack of a secure element, brute-force attacks on the user's passcode can be executed off-device and speed up by utilising external processing resources.

References

1. Apple: iOS Security. Tech. Rep. May, Apple Inc. (2012), http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf
2. Belenko, A., Sklyarov, D.: Evolution of iOS Data Protection and iPhone Forensics : from iPhone OS to iOS 5 (2011)
3. Chen, Y.C.Y., Ku, W.S.K.W.S.: Self-Encryption Scheme for Data Security in Mobile Devices (2009), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4784733
4. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android Security (2009), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4768655
5. Goadrich, M.H., Rogers, M.P.: Smart Smartphone Development : iOS versus Android. Science Education pp. 607–612 (2011), http://dl.acm.org/citation.cfm?id=1953330
6. Heider, J., Khayari, R.E.: iOS Keychain Weakness FAQ - Further Information on iOS Password Protection (2012), http://sit.sit.fraunhofer.de/studies/en/sc-iphone-passwords-faq.pdf
7. Kaliski, B.: PKCS #5: Password-Based Cryptography Specification Version 2.0 (2000), http://www.ietf.org/rfc/rfc2898.txt
8. Pacatilu, P.: Android Applications Security. Informatica Economica 15(3), 163–171 (2011), http://search.ebscohost.com/login.aspx?direct=true\&db=bth\&AN=69706020\&site=ehost-live
9. Pandya, V.R.: IPHONE SECURITY ANALYSIS. Journal of Information Security 1(May), 74–87 (2008), http://www.scirp.org/journal/PaperDownload.aspx?DOI=10.4236/jis.2010.12009
10. Paul, M., Chauhan, N.S., Saxena, A.: A security analysis of smartphone data flow and feasible solutions for lawful interception (2011), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6122788
11. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., Glezer, C.: Google Android: A Comprehensive Security Assessment (2010), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5396322
12. Shurui, L.S.L., Jie, L.J.L., Ru, Z.R.Z., Cong, W.C.W.: A Modified AES Algorithm for the Platform of Smartphone (2010), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5636941
13. Zovi, D.A.D.: Apple iOS 4 Security Evaluation (2011), http://www.trailofbits.com/resources/ios4_security_evaluation_paper.pdf