An Overview About the Security Architecture of the Mobile Operating System iOS

** Jithu Philip*
*** Merin Raju*

Abstract

The use of mobile operating systems is rapidly increasing as the number of smartphone users is increasing day by day. iOS is one of the most popular and widely used mobile operating systems in the world. This document describes an overview about the security features of iOS in its low-level system framework and in its user-level applications framework. The document discusses processes like how security is implemented in the core components of the system, how data protection and encryption is achieved, and how applications are secured by sandboxing.

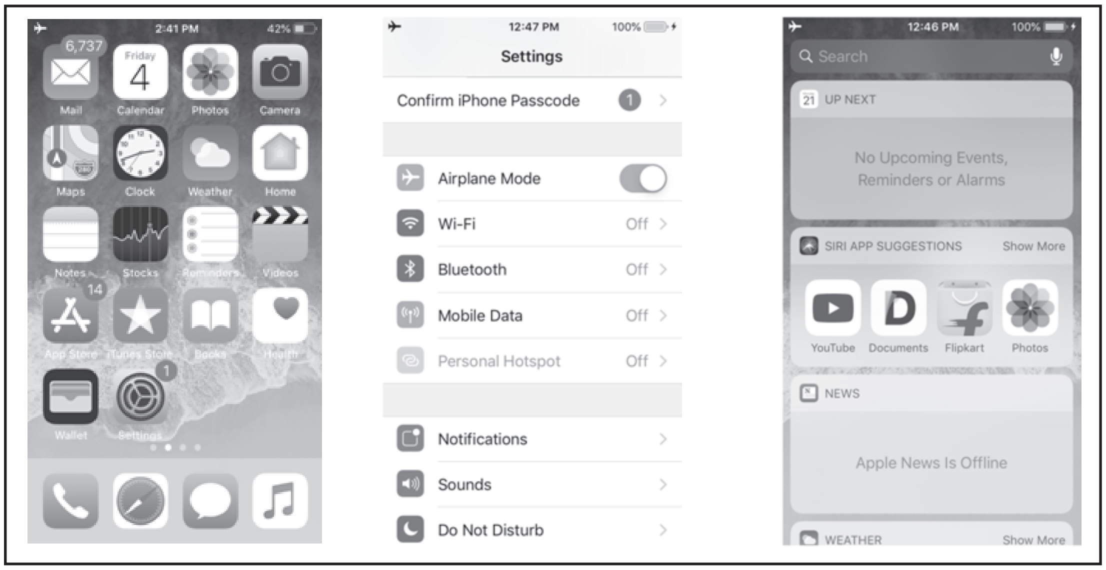
Keywords: Apple, application security, data security, encryption, iOS, operating systems, system security

I. INTRODUCTION

iOS is a mobile operating system created and developed by Apple Inc., specifically for its mobile hardware. The initial version of iOS was released in 2007 for the iPhone. Apple later expanded its support to other Apple's own portable hardware devices like iPad and iPod Touch.

II. GENERAL OVERVIEW

The interface of iOS is designed in a user friendly manner, based upon direct manipulation using multi-touch gestures. A demonstration of how the user interface of iOS looks like is given in Fig. 1.



12:47 PM 100%

Settings

Confirm iPhone Passcode 1

Airplane Mode

Wi-Fi Off

Bluetooth Off

Mobile Data Off

Personal Hotspot Off

Notifications

Sounds

Do Not Disturb

Fig. 1. iOS 12 Running on an iPhone SE

Manuscript received November 2, 2018; revised November 25, 2018; accepted December 2, 2018. Date of publication January 6, 2019.
* J. Philip is an independent security researcher. (email: jithuphiliph@gmail.com)
** M. Raju is Lecturer (Computer Science) with Department of Commerce, Bishop Kurialacherry College for Women, Amalagiri, Kottayam, Kerala, India-686561. (email: merin.raju12s@gmail.com)
DOI:10.17010/ijcs/2019/v4/i1/142412

A. Hardware

The hardware that iOS runs is of the Advanced RISC Machine (ARM) Architecture [13], [14]. iOS versions released before iOS 7 can only support devices with 32-bit ARM processors (ARMv6 and ARMv7-A architectures). From iOS 11 onwards Apple drops support for 32-bit ARM processors and 32-bit applications, thereby making iOS a 64-bit only Operating System.

B. Kernel

The iOS uses the same UNIX based XNU core [10] of Darwin. The initial version of iOS, that is, iOS 1.0 used the Darwin 9.0.0d1. The current version of iOS that is iOS 12 is based on Darwin 18.

C. Software Updates

Devices running iOS can be updated to latest versions through OTA update or through iTunes update service. iOS specifically uses a process called *System Software Authorization* to prevent devices from being downgraded to previous iOS versions that lack the latest security updates and are vulnerable to attacks. The boot process in an iOS device ensures that only the code signed by Apple can be installed on the device [1].

III. DESIGN OF THE LOW-LEVEL SYSTEM ARCHITECTURE

The design of iOS is done in such a way that the integration between hardware and software is secure across all the core components. This goal is achieved through various levels of integrity check done within the encrypted hardware and the signed software, developed by Apple. The Security Architecture diagram of iOS [2], [3] is shown in Fig. 2.

A. The Boot Process (Secure boot chain)

The boot process contains components that are cryptographically signed by Apple, which verifies the integrity of each component before proceeding to the next, thereby accomplishing a secure boot chain.

When an iOS device is turned on, its *application processor* executes code from the read only memory called the BOOT ROM. The executable code contained in BOOT ROM is written during the chip fabrication stage and hence, it can be implicitly trusted. The BOOT ROM contains the Apple Root CA public key which verifies whether the bootloader (iBoot) is signed by Apple before proceeding to execute it.

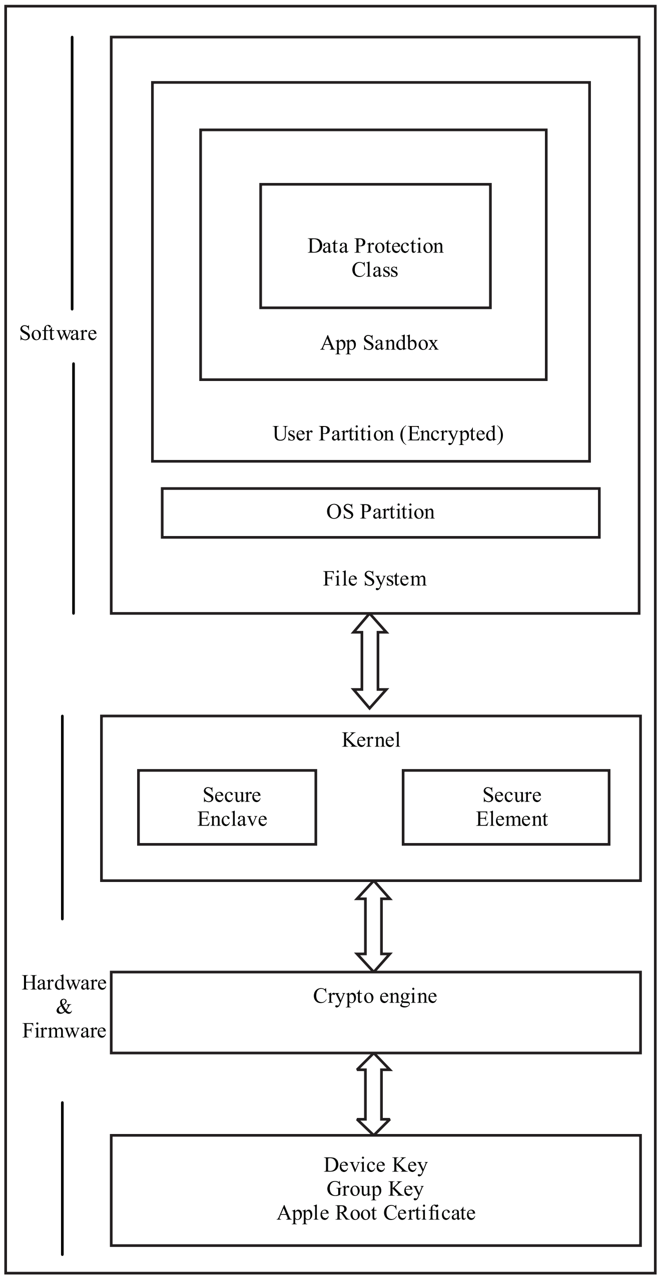


Fig. 2. Security Architecture Diagram of iOS

On older devices with A9 chip or earlier A-series processors, an additional bootloader called the Low-Level Bootloader (LLB) is loaded before iBOOT. Upon the successful execution of the iBOOT, the system kernel itself is verified and executed by the iBOOT.

A failure in BOOT ROM to load LLB (on older devices) or iBOOT (on newer devices) causes the system to enter DFU (Device Firmware Update) mode and a failure in LLB or iBOOT to verify its next stage causes the system to enter recovery mode.

B. Secure Enclave

Secure Enclave is a coprocessor that is used in iOS device's system on chip (SOC). It uses encrypted memory and does all cryptographic operations for data protection [9]. During hardware fabrication, each device's secure enclave is assigned a unique ID, which cannot be changed. This unique ID is used in the process of temporary key generation for memory encryption. The secure enclave also contains a hardware random number generator as a part of the coprocessor.

Secure Enclave contains a dedicated Secure Enclave Boot ROM like the application processor Boot ROM, which creates an ephemeral memory protection key at system startup. This key in combination with the device's UID is used for the encryption of Secure Enclave's portion of the device's memory.

Data that is written to the file system by Secure Enclave is further encrypted with a key bound with the UID and an anti-replay counter (which is stored in a dedicated nonvolatile Integrated Circuit (IC).

The processing of fingerprint and face data from Touch ID and Face ID sensors for user authentication is also done by the Secure Enclave.

C. Hardware Encryption

For ensuring higher level of security, encryption stands as a mandatory requirement for every stage in an iOS operation. A major use of encryption in iOS is in the memory of the Secure Enclave. iOS devices use AES-256 cryptographic encryption that secures the DMA path between flash storage and system main memory [1] [9].

iOS with its hardware encryption makes sure that the security of the device is not compromised in any level of its operation. If a user erases the contents and settings of an iOS device, it then deletes all the keys to the encrypted storage, thereby making the data on the device cryptographically inaccessible. iOS uses crypto-shredding to achieve this.

IV. DATA PROTECTION

iOS devices protect data stored in its flash storage by constructing and managing a hierarchy of keys in combination with the employed hardware encryption.

This technique is done on a per-file basis, where each file is assigned to a class, and the file can be accessed only based on whether the key to that class is unlocked or not.

Newer version of iOS file system, APFS (Apple File System), sub-divides the key to a per-extent basis (portions of a file can hold multiple keys) [12]. The Data Protection and Encryption process of iOS [2], [3] is shown in Fig. 3 .

A. How the Data Protection Architecture Works?

When a new file is created on the data partition, the protection mechanism creates a 256-bit key (per-file key), which is then forwarded to the hardware AES engine for the encryption of the created file on the system's flash storage.

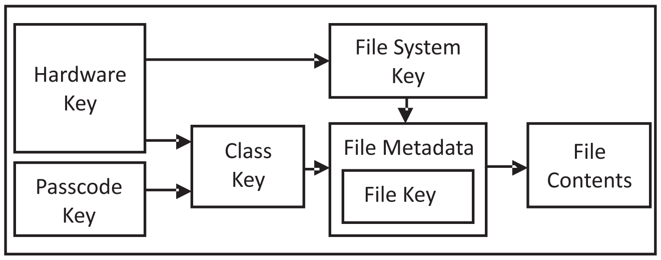


Fig. 3. Data Protection and Encryption Process of iOS

The per-file or per-extent key is wrapped with one of several class keys and stored in the file's metadata for further usability, depending on how the file is accessed.

While opening a file from the encrypted storage, the file's metadata is decrypted with the file system key, thereby revealing the wrapped per-file key and the class key. The per-file (or per-extent) key is unwrapped with the class key and given to the hardware AES engine for the decryption of the exact file contents from the storage. All wrapped file key handling operations are done by the Secure Enclave and the keys are never directly exposed to the application processor.

The metadata of all files is encrypted with a random key (which itself is created when iOS is first installed or a factory reset operation is done by the user). For devices having APFS (Apple File System), the wrapping of file system metadata key is done by the Secure Enclave UID (Unique ID which was stored when the chip fabrication was done) for long-term storage.

B. Advances of Data Storage in APFS With Per-extent Keys

The introduction of Apple File System (APFS) [12] provides advanced data storage capabilities by cloning of files, which uses copy-on-write technology, and reduces the actual cost of copy operation [7], [8]. A clone creates a copy of a file or directory with no additional space on the disk. In this technique both the file and its clone share the same set of unmodified blocks. Data blocks which are modified only are written elsewhere in separate blocks.

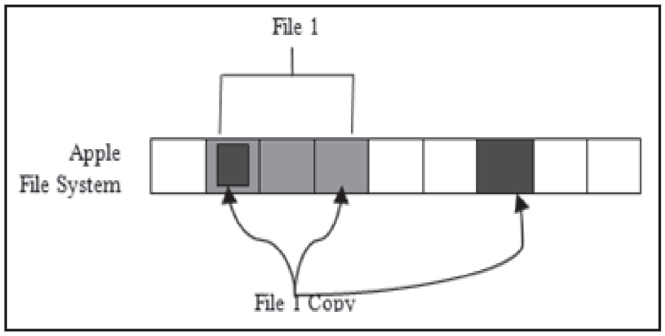


Fig. 4. Storage representation of a how file and its clone share blocks in Apple File System

The demonstration of a file and its clone, each of which shares two common blocks and one separate block stored in an Apple File System is given in Fig. 4. For this representation to work, if a file is cloned, each half of the clone gets a new key. So modifications to the file or its clone correspond to a new key. Over time, the actual file and its clone's modifications correspond to different extents or fragments, each of which is represented by a different key. Anyhow, all extends of a file will be managed by the same class key.

V. APPLICATION SECURITY IN APPLE FILE SYSTEM (APFS)

For security purposes, the file system of iOS is treated in such a way that users do not have direct access to the file system. This design strategy makes an iOS application only access the directories which reside inside its own sandbox directory. For that to be achieved, the installer creates different container directories inside an applications' sandbox directory during the installation of the app. The container directories that were created are meant for different purposes. The bundle container directory holds the app's bundle, and the data container directory holds the data for both, the app and the user. The data container directory can create further sub directories within itself so that data can be organized. A representation of an iOS application operating within its own sandbox directory [8] is shown in Fig. 5.

In general, an application installed in iOS is restricted from accessing or creating files outside its container directory. An exception to this kind of access restriction can be achieved for doing some specific tasks with the help of public system interfaces (for example, an application is allowed access to the users' contacts). More details regarding the security effect of jail-breaking an iOS device can be found in [5] for study.

VI. NETWORK SECURITY

iOS supports Transport Layer Security [9] (TLS v1.0, TLS v1.1, TLS v1.2), and DTLS. It supports both AES-128 and AES-256, and prefers cipher suites with perfect forward secrecy. TLS with both low-level and high-level APIs are supported for developers to make use of it in their apps. As web security is a fact that mobile operating systems needs to accomplish through security patches everyday, experiments related to that are relevant for researchers and can be found in different documents for study [4-6].

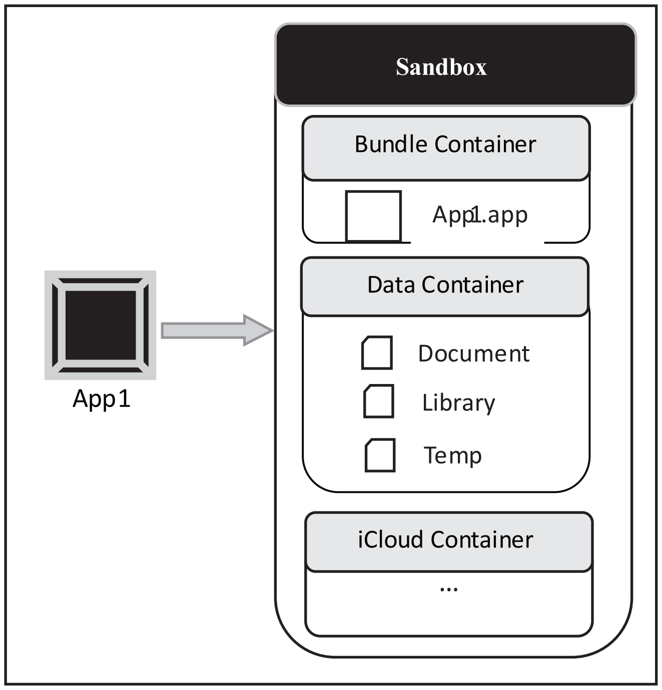


Fig. 5. An iOS application operating within its own sandbox directory

VII. PROTECTION OF RESOURCES AND SYSTEM PROCESSES AT RUNTIME

iOS protects the core components of the system by limiting access to its system files and resources. The entire operating system partition is mounted as read-only for user operations. The majority of tasks in iOS are performed in a non-privileged user mode. Services that are unnecessary are not included in the system, for example, remote login services. Escalation of privileges done by applications through the use of APIs that modify other applications or the operating system itself is restricted. For privileged operations to be done, iOS makes use of signed entitlements. Entitlements are digitally signed entities that cannot be changed and are used by system apps and daemons to perform specific privileged operations.

A. Address Space Layout Randomization (ASLR)

iOS makes use of Address Space Layout Randomization (ASLR) [2], [11], a low-level technique that prevents against exploitation of memory corruption bugs. ASLR ensures that all memory regions are randomized on execution. Data is placed in randomly selected locations in memory that reduces ways to create exploits to corrupt the system.

B. Execute Never (XN)

iOS uses ARM's Execute Never (XN) feature [2], [11], which marks memory pages as non-executable. This makes apps to only use the portions of the memory that are marked as both writable and executable.

VIII. USER AUTHENTICATION MEASURES

Touch ID, Face ID, and passcodes are the measures that a user can enable in order to secure an iOS device. To use biometric access measures like Touch ID or Face ID, the device needs to be set so that a passcode is required to unlock it. Data protection is automatically enabled when a passcode is set. The processing of fingerprint and face data from Touch ID and Face ID sensors for user authentication is done by the dedicated coprocessor, Secure Enclave.

IX. CONCLUSION

As described in this document, iOS is a secure mobile operating system which ensures security as a core feature in its workflow. It ensures a secure signing procedure so that the boot code cannot be tampered with malicious code. The data stored in an iOS device is encrypted with a 256 bit AES cryptographic engine. The coprocessor Secure Enclave takes care of the biometric authentication securely, and it also does the temporary key generation operation for memory encryption, with its Unique ID. Applications running in iOS are secured by sandboxing each application's directories from others. Protection of resources and system process are made at runtime by techniques like ASLR and XN. With all these measures made available, the design strategy of iOS makes it one of the most secure mobile Operating Systems in the world.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable and insightful comments on the details of this document.

REFERENCES

[1] iOS Security Overview, 2018. [Online]. Available: https://www.apple.com/business/resources/docs/iOS_Security_Overview.pdf

[2] iOS Security Guide, 2018. [Online]. Available:

https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf

[3] B. Guo, "iOS Security," 2014. [Online]. Available: https://www.cse.wustl.edu/~jain/cse571-14/ftp/ios_security/index.html

[4] F. Al-Qershi, M. Al-Qurishi, S. M. M. Rahman and A. Al¬-Amri, "Android vs. iOS: The security battle," In *2014 World Congr. on Comp. Appl. and Inform. Syst. (WCCAIS)*, pp. 1¬8. IEEE, 17-19 Jan, 2014. doi: 10.1109/WCCAIS.2014.6916629

[5] T. Wang, K. Lu, L. Lu, S. Chung and W. Lee. "Jekyll on iOS: When benign apps become evil," in *22nd Usenix Security Symp.*, August 14-16, 2013, pp. 559¬572. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/wang_tielei

[6] R. Wang, L. Xing, X. F. Wang and S. Chen. "Unauthorized origin crossing on mobile platforms: Threats and mitigation," In *Proc. of the 2013 ACM*

SIGSAC Conf. on Comp. & Commun. Security, pp. 635¬646. ACM, 2013. [Online]. Available: https://www.informatics.indiana.edu/xw7/papers/wang2013unauthorized.pdf

[7] About Apple File System. [Online]. Available: https://developer.apple.com/documentation/foundation/file_system/about_apple_file_system

[8] Apple File System Programming Guide. [Online]. Available: https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html

[9] J. R. Vacca, *Network and System Security*, Syngress, 2010.

[10] M. J. Bach, *The Design of the Unix Operating System*, Pearson Edu. Inc., 1986.

[11] iOS. [Online]. Available: https://en.wikipedia.org/wiki/IOS

[12] Apple File System. [Online]. Available: https://en.wikipedia.org/wiki/Apple_File_System

[13] ARM Architecture. [Online]. Available: https://en.wikipedia.org/wiki/ARM_architecture

[14] RISC. [Online]. Available: https://en.wikipedia.org/wiki/Reduced_instruction_set_computer

About the Authors

Jithu Philip received M. Sc. degree in Computer Science from School of Computer Sciences, Mahatma Gandhi University, Kottayam, Kerala in 2014. He is currently working as Multimedia Specialist for Philco Media, Kottayam, Kerala. He is also focused on conducting simulations as an independent security researcher and has research papers related to Operating Systems, Computer Security and Image Processing with specific focus on image forensics published in various journals.

Merin Raju received M.Sc. degree in Computer Science from School of Computer Sciences, Mahatma Gandhi University, Kottayam, Kerala in 2014. She is working as Lecturer (Computer Science), Department of Commerce, Bishop Kurialacherry College for Women, Amalagiri, Kottayam, Kerala. Her research interests are in the areas of Computer Security and Image Processing.