Mathematical Problems of Computer Science 50, 88--95, 2018.

Secure Storage Implementation for Large Files in iOS Environment

Levon M. Hovsepyan and Aren K. Mayilyan

National Polytechnic University of Armenia
e-mail: lehovsepyan@gmail.com, mayilyan96@mail.ru

Abstract

Mobile device storage services are not secure by nature as all databases and files are stored on the client side. We cannot ignore that because nowadays mobile devices are made of hardware, which is capable for building and running applications, which cover almost every functionality that computers have. There is an inherent risk of data exposure (confidentiality) and data tampering (integrity). To avoid the risks mentioned above, software engineers use some approaches, which are provided by the iOS SDK for securely storing sensitive data. However, those approaches currently work only for small amount of data (key-value pairs) and the issue still remains for large files.

In this paper, we introduced an approach of securely storing large files in iOS environment.

Keywords: iOS, Secure storage, Data protection, Encryption

1. Introduction

Nowadays people use smartphones more often than computers in daily life. Researches show that in 2016, an estimated 62.9 percent of the population worldwide already owned mobile phones [1], and in 2019 the number of mobile phone users is forecast to reach 4.68 billion. The tendency towards mobile devices leads to technological progress and hardware technologies already reached a level that giant manufacturers like Apple, Samsung, Xiaomi produce smartphones with technical characteristics capable to compete with computers by their performance and run heavy CPU/GPU intensive applications like different games and AI powered apps.

This mobile revolution brought almost every sphere to a digital platform, which raised data protection and privacy risks. For example, a lot of personal information, photos, videos and

other files are now being published on social platforms, and it's an urgent task for the software company to protect that data and make it accessible only for specified people. Another example is that people store their passwords inside one software for remembering them easily, and the result of data leak here can be irreversible. Those two examples mentioned above highlight the importance of data protection.

In enterprise solutions, data protection is relatively easy than in mobile ones. In enterprise systems, data is being stored on server side, inside the specially designed databases or even data centers. This approach is more secure in contrast to mobile solutions for the following reasons:

- Server side hardware has more computational power and memory storage, and in addition to it, both of these parameters can be increased any time.
- The access to server side storage is happening through network, and many penetration attempts can be blocked at network layer.
- Server side data storage can be isolated from network at any time to reduce the access

In mobile platform, data is stored in embedded memory of mobile device, which gives full control to an intruder over it. Even though iOS has a secure file system divided into sandboxes [2], as shown in Figure 1, it becomes accessible after device jailbreak. This means that the data can be observed as it is from the file directories. The only data protection method that can work here is to store everything in encrypted form, which will prevent data from being leaked even if it is accessible.

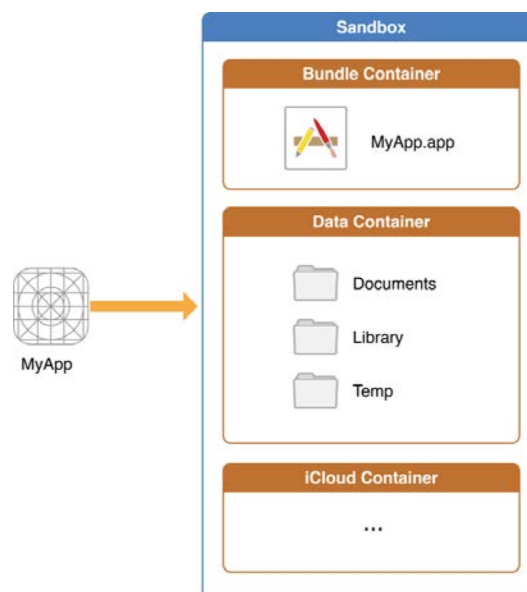

Fig.1 . An iOS app operating within its own sandbox directory.

As a solution, iOS platform gives Keychain [3] to developers, which is a shared key-value secure storage. It works fine for small data like passwords, cryptographic keys, certificates and notes, but when it comes to larger files, keychain becomes problematic to use. As it is shared

storage between devices, which are connected to the same Apple Id, it syncs the stored files to iCloud storage. For large files it will work slow and can cost a lot of money for network traffic.

In this paper, we introduce an approach for securely storing large files in local memory. As the mobile devices have limitations in memory, the approach supports incremental usage as well. The security will be based on AES-256 encryption algorithm, and the encryption key will be securely stored in keychain storage.

2. Related Work

As data protection is an actively studied subject, there are many suggested solutions in regard to how to securely store the data. Particularly for iOS platform, the most common solutions for storing data are Core Data and Keychain.

Core Data is Apple's persistence framework with an underlying SQLite database. This means that developers interact with Core Data methods, not the database directly. An important observation is that SQLite is not encrypted by default, when the device is unlocked. Nevertheless, Apple has a feature called "Data Protection" [4], which encrypts the sandbox while the device is locked with passcode. However, we cannot rely on the end user to passcode-protect his/her device. The device can be jailbroken and passcodes can be easy to crack. Thus, we see that this solution has vulnerabilities and contains risks for storing sensitive data.

The second secure option that iOS SDK gives is Keychain. Keychain API's solve secure storage problems by giving the app a mechanism to store small bits of user data in an encrypted database.

The keychain is not limited to passwords, as shown in Figure 2. You can store other secrets that the user explicitly cares about, such as credit card information or even short notes. It is also a good place for storing items that the user needs, but may not be aware of, e.g., the cryptographic keys and certificates.

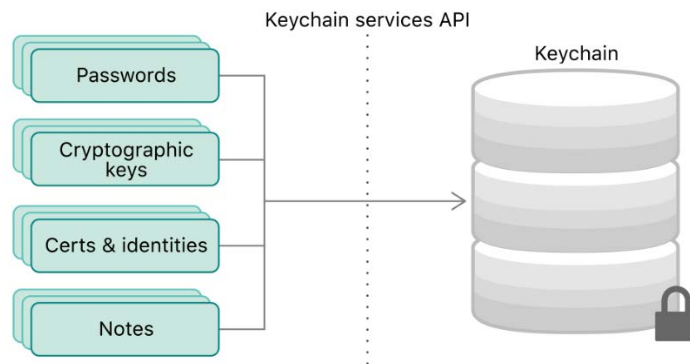

Fig. 2. Securing the user's secrets in a keychain.

Keychain items are encrypted using two different AES-256-GCM keys: a table key (metadata) and per-row key (secret key). As a metadata, "created" and "last updated" timestamps are being generated, when storing data. Keychain metadata is encrypted with the metadata key to speed-search, while searching secret value is encrypted with the secret-key.

The Keychain is implemented as a SQLite database stored in the file system. There is only one database, and the "securityId" daemon determines which Keychain items each process or app can access. Keychain items can only be shared between apps from the same developer. This is managed by requiring third-party apps to use access groups with a prefix allocated to them

through the Apple Developer Program through application groups. The prefix requirement and application group uniqueness are enforced through code signing [5], provisioning profiles and Apple Developer Program.

Even though Keychain is a securely designed and easy-to-use solution: its disadvantage is the incapability to store large files.

3. Overview of Our Approach

As we saw in the previous section, native approaches have some disadvantages, and it is reasonable to come up with a solution, which will cover them. Firstly, we need to choose the correct encryption algorithm. The most important characteristic to keep in mind when choosing an encryption algorithm is that you should select one that is widely used and accepted by the security community. The security of any encryption algorithm should never depend upon the secrecy of the algorithm itself. Instead, the algorithm should be publicly disclosed and open to the cryptographic community for analysis. The true security of the algorithm always lies in the security of the keys used to decrypt message. After some researches and experiments AES-256 with CBC mode was chosen as the encryption algorithm. AES is Advanced Encryption Standard, symmetric block encryption mechanism. 256 in AES-256 is a key size in bits.
It is based on a design principle known as a substitution-permutation network, and is efficient in both software and hardware. AES-256 is a variant of Rijndael, which has a fixed block size of 128 bits, and a key size of 256 bits. This cypher is considered among the tops and is widely used in SSL/TLS over the Internet. The only weakness of this algorithm is the key that we choose. As long as we choose a strong key for it, AES-256 will keep our files safe.

As we are developing an approach for mobile side encryption, we consider that there is no way to retrieve encryption key from outside the phone. Hence, the method of storing the key itself becomes a vulnerable point. If we hardcode the encryption key inside our source code, it can be easily hacked after jailbreak using reverse engineering [6] techniques. To avoid that issue, we suggest to generate an encryption key based on the advertising identifier (IDFA), which is a unique ID for each iOS device's application that mobile ad networks typically use to serve targeted ads. This unique id will not be visible in the source code even when you reverse engineer the app, therefore it will prevent the secret key exposure. For key generation, we use PBKDF2, which is a key derivation function [7], which derives a secret key from IDFA using a pseudorandom function [8]. After successful generation, we store the generated secret key in the keychain where we can be sure it is safe.

Now when the secret key is generated, we are ready to encrypt any file and store it inside the "Documents" directory of the application. There are various open-source cryptographic algorithm implementations available to use in iOS, which are mostly based on Apple's Corecrypto library. For our approach we chose RNCryptor [9], because it supports incremental use [10]. Incremental use of encryption is useful for cases where the data will no comfortably fit in memory, which is a common situation for mobile devices, or when the data is received via Internet with chunks. Receiving and encrypting data chunks asynchronously saves time. The overall process of file encryption is presented in Figure 3.

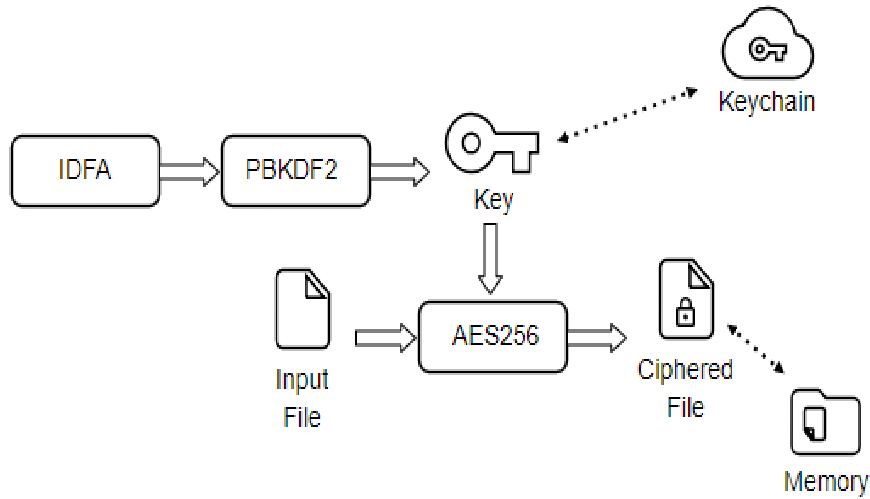

Fig. 3. Overall process of file encryption.

As the approach is designed for large files, it is important to have acceptable performance. Performance measurements have been done on iPhone 7, which has an average computing power. Encryption experiments for relatively large files showed acceptable performance, as you can see in Figure 4.

The overall performance of data storage is presented in Figure 5, which shows a comparison of data storing with and without encryption. The results can vary on different devices based on CPU power, but in average they are acceptable.

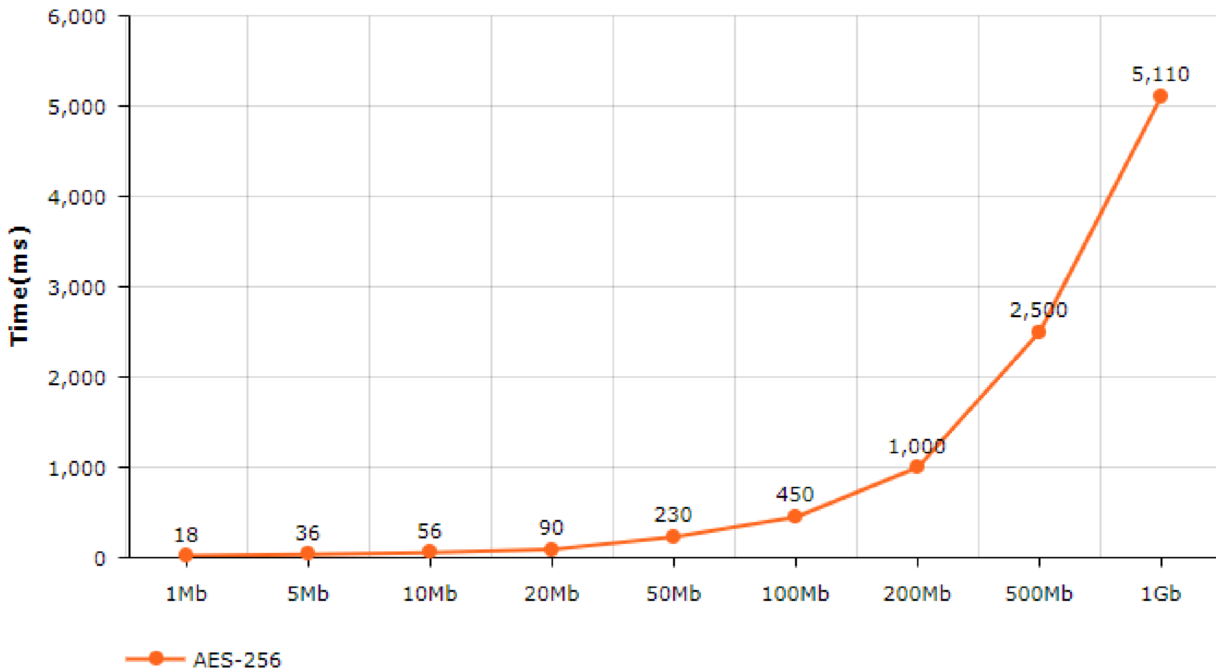

Fig. 4. File size/Encryption time measurements.

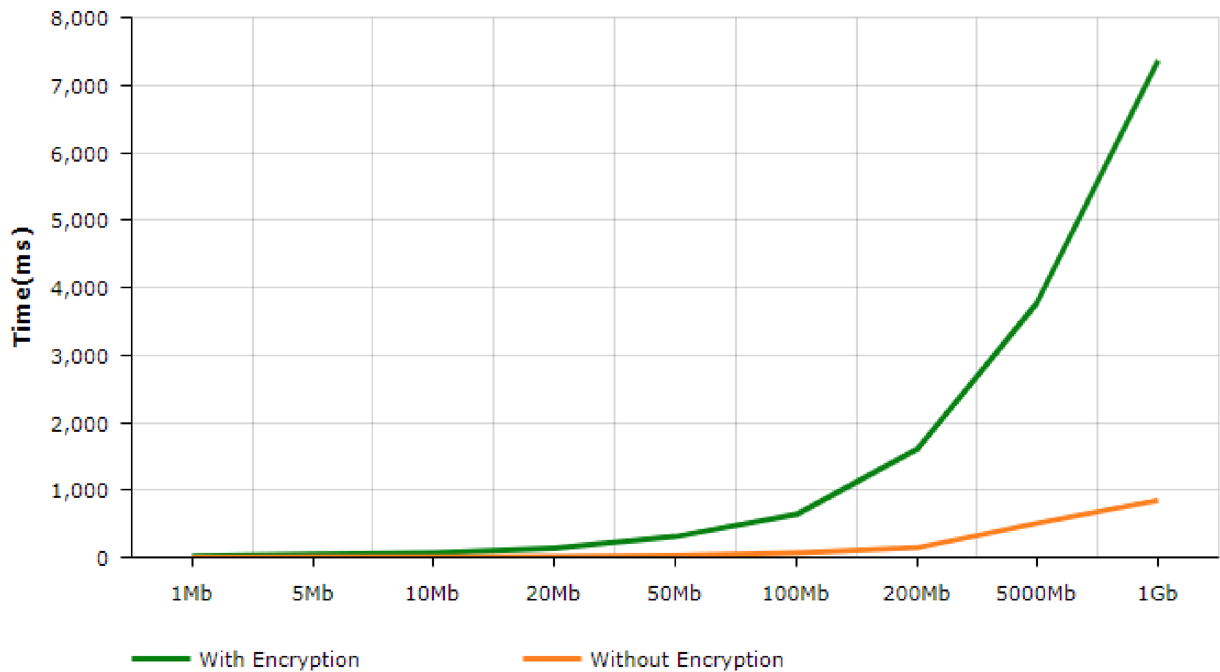

Fig. 5. File storage performance with/without encryption.

4. Conclusion

In this paper, we have shown an approach for securely storing large files in local memory of an iOS device using cryptographic algorithms. We have shown how to generate the encryption key to protect it in case the application is reverse-engineered. This approach differs from the existing ones, because it still protects your data, even when the device is jailbroken and the access to applications file system is open to anyone. Moreover, this approach increases the performance of data storage with the help of incremental encryption.

References

[1] [Online]. Available: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/
[2] [Online]. Available: https://cocoacasts.com/what-is-application-sandboxing
[3] Apple's iOS Security Guide, pp 20-22, 2018
[4] https://developer.apple.com/documentation/uikit/core_app/protecting_the_user_s_privac
[5] Code Signing - The Internet Protocol Journal, Volume 5, Number 1 *by Eric Fleischman*
[6] E. Mona and M. Ali, "Reverse Engineering iOS Mobile Applications", *Proceedings-Working Conference on Reverse Engineering, WCRE,* 10.1109/WCRE.2012.27, pp. 177-186, 2012.
[7] C. Adams, G. Kramer, S. Mister and R. Zuccherato, "On the security of key derivation functions", *7th Internation Conference on Information Security*, *Lecture Notes in Computer Science*, vol. 3225, pp. 134-145, 2004.

[8] J. Hastad, R. Impagliazzo, L. Levin and M. Luby, “A pseudorandom generator from any one-way function”, *SIAM Journal on Computing*. 28. 10.1137/S0097539793244708, vol.-28, no. 4, pp. 1364-1396, 1999.
[9] [Online]. Available: https://github.com/RNCryptor/RNCryptor
[10] I. Mironov, O. Pandey, O. Reingold, and G. Segev, “Incremental Deterministic Public-Key Encryption”, *Journal of Cryptology*. 31. 10.1007/s00145-017-9252, vol.31, no. 1, pp. 134-161, 2017.

Submitted 05.06.2018, accepted 22.11.2018.

iOS Միջավայրում մեծ ֆայլերի ապահով պահպանման մեթոդի իրականացում

Լ. Հովսեփյան և Ա. Մայիլյան

Ամփոփում

Բջջային սարքավորումներում տվյալների պահպանման ծառայություններն ապահով չեն իրենց բնույթով, քանի որ բոլոր տվյալների հենքերը և ֆայլերը պահպանվում են օգտագործողի կողմում: Մենք չենք կարող այս երևույթից խուսափել, որովհետև մեր օրերում բջջային սարքավորումները պատրաստվում են այնպիսի ապարատային սարքավորումներից, որոնք ի զորու են կառուցել և աշխատեցնել հավելվածներ, որոնք ծածկում են համակարգչին բնորոշ գրեթե բոլոր ֆունկցիոնալ հնարավորությունները, և շատ մեծ կիրառություն ունեն: Ապահով չլինելուց բխում են տվյալների բացահայտման (գաղտնիություն) և խարդախությամբ նրանց մեջ միջամտման (ամբողջականության) ռիսկեր: Վերոնշյալ ռիսկերից խուսափելու համար, ծրագրային ճարտարագետներն օգտագործում են iOS SDK-ի կողմից տրամադրվող մոտեցումներ, նշանակալի կարևորության ֆայլերի ապահով պահպանման համար: Այնուամենայնիվ, այդ մոտեցումները ներկա պահին կիրառելի են միայն փոքր ծավալի տվյալների (բանալի-արժեք) դեպքում, և խնդիրը դեռևս առկա է մեծ ֆայլերի դեպքում:

Այս հոդվածում մենք առաջարկում ենք մոտեցում, iOS միջավայրում մեծ ֆայլերի ապահով պահպանման համար:

Реализация безопасного способа хранения больших файлов в iOS среде

Л. Овсепян и А. Маилян

Аннотация

Сервисы хранения данных для мобильных устройств по натуре не зашищены, так как все базы данных и файлы хранятся в стороне пользователя. Мы не можем обойти это, потому что в наше время мобильные устройства изготовлены из апаратных средств, которые способны строить и привести в работу приложение, которые в себе содержат весь функционал, что имеет компьютер, и имеют массовое примемение. От этого существует неотъемлемый риск экспозиции (конфиденциальность) и фальсификации данных (целостность). Чтобы избежать упомянутых выше рисков, разработчики используют подходы предоставляемые iOS SDK для безопасного хранения конфиденциальных данных. Тем не менее, эти подходы в насоящее время работают только для небольшого количества данных (пары ключ-значение), и проблема остается для больших файлов.

В этой статье мы представили подход безопасного хранения больших файлов в среде iOS.