A Study on Symmetric Encryption Mechanism with Fingerprint Recognition – a Case of Android Smartphone

Chen-Hua Fu[1*] and Chen-Ming Yeh[2]

[1]*Department of Information Management, College of Management, National Defense University*
[2]*Information, Communications and Electronic Force Command*

ABSTRACT

This study attempted to use the built-in fingerprint device on the smart mobile device and symmetric encryption mechanism to assure the security of the stored files in the smart mobile device. The encryption mechanism proposed by this study integrates the personal fingerprint verification function and AES256 block encryption mechanism. The proposed encryption mechanism generates the encryption key dynamically based on the name and length of the plaintext file. With dynamic secret key generation mechanism, the proposed encryption mechanism can enhance the difficulty of decryption of encrypted files and effectively improve the convenience and availability of encrypted files. Finally, this study used Android Studio as App development platform to develop an App with Java programming language. The App calls encryption core program written in C codes with JNI to complete an encryption app that can be implemented in an Android smart mobile device. With several scenarios, the results showed that the feasibility and practicability of the proposed encryption mechanism applied to Android smart mobile devices were verified.

Keywords: Smart mobile device, Fingerprint detection, Symmetric encryption, Dynamic secret key

應用指紋辨識於對稱式加密機制之研究 —以 Android 智慧型手機為例

傅振華[1*] 葉乘銘[2]

[1]國防大學管理學院資訊管理學系
[2]資通電軍指揮部

摘　要

本研究嘗試運用智慧型行動裝置上內建指紋裝置及對稱式加密機制確保所儲存機敏文件的安全性；研究所提加密機制將整合個人指紋辨識驗證與 AES256 區塊加密機制，透過檔案長度及名稱動態產生加密密鑰，以強化密文資料被破解的難度，以有效提升加密資料使用的便利性及可利用性。最後，本研究以 Android Studio 做為研究應用程式開發平臺，透過 Java 程式語言開發 App，並利用 JNI 呼叫以 C 程式語言所撰寫的加密核心程式，完成一個 Android 智慧型行動裝置適用的 App，並進行實驗驗證，其結果驗證本研究所提加密機制應用於 Android 智慧型行動裝置之可行性與實用性。

關鍵詞：智慧型行動裝置、指紋辨識、對稱式加密、動態密鑰

文稿收件日期 110.10.13; 文稿修正後接受日期 111.2.14; *通訊作者
Manuscript received October 13, 2021; revised February 14, 2022; * Corresponding author

I. INTRODUCTION

In recent years, in response to the popularity of smart mobile devices, the number of Android users worldwide has exceeded 2 billion. It is usual for employees to carry their smartphones, tablets, notebook computers, and other devices at offices. Employees can access customer data, print reports, and reply to the boss's instructions anytime and anywhere; this improves their work productivity efficiently. However, for an enterprise, employees might be the biggest threat to the enterprise's data [1, 2]. However, when employees lose their smart mobile devices or do not take security measures to protect data in their smart mobile devices. The enterprise may leak critical data from its employees' smart mobile devices.

Each individual has unique national features, such as eye coloration, fingerprint, palm print, voice, facial image, and DNA signature. A computer can perform some authentications with those national features [3]. Biometric identification technology, such as fingerprint recognition or face identification, has been developed as an additional authentication way to enhance information security and network security [4]. Fingerprint-based biometrics will cover the traditional password-based and token-based authentication aspects to improve system security [5].

This study will propose a refined symmetric encryption mechanism with fingerprint identification on smartphones to enhance the security of encryption and decryption procedures. With the proposed symmetric encryption mechanism, we can improve the data's confidentiality and increase the data's availability on a smart mobile device. For protecting data's confidentiality, this study bases on symmetry encryption mechanism and access control concept to propose a symmetry block encryption mechanism, which fingerprint recognition is adopted. For implementing the proposed encryption mechanism on Android smartphones, this study used Android studio to develop the proposed encryption mechanism; this study also called JNI (Java Native Interface) [6] in Android studio to Integrate the AES encryption program. The developed encryption app can improve users' data confidentiality on their smart mobile devices. Moreover, since the proposed encryption adopts a fingerprint recognition measure, users can decrypt an encrypted file only with a correct fingerprint. The fingerprint recognition measure implements an access control mechanism in the proposed encryption mechanism. Therefore, the proposed encryption mechanism can the confidentiality and access control mechanism for the data on a smart mobile device.

II. RELATED WORKS

We discuss several topics related to the proposed encryption mechanism in the following subsections.

2.1 Fingerprint recognition

Usually, an identification system based on biometrics can identify persons with their physiological characteristics [7]; fingerprint recognition is one of the biometric identification systems. Each person has unique fingerprints; we identify a specific person based on fingerprint recognition. Identifying one person by his fingerprint is like confirming the caller in network communication based on MAC [8].

A pattern of ridges and valleys covers the skin on the inside of a finger. Every person has a unique fingerprint, which is different from any other person. The fingerprint identification depends on two basic assumptions: - Invariance and Singularity Invariance, which means the fingerprint characteristics do not change along with the life [9]. Many studies have explored whether these patterns were different for each person several centuries ago, and indeed every person is believed to have unique fingerprints [10]. This study result makes fingerprints suitable to identify the identification of their owner [7].

Fingerprints are graphical patterns of ridges and valleys on the surface of fingertips; the ridge ending and ridge bifurcation is called minutiae. The minutiae are the points where the ridge structure changes, such as bifurcation and endpoint [9]. Fig. 1 shows the different minutiae. For fingerprint recognition systems, most systems only use specific features in the pattern.

These features are since the papillary ridge in the fingerprint pattern is not a continuous line; but a line at the end, which splits into forks (called forks) or forms islands. Although the fingerprint usually contains about 100 minutiae, the fingerprint area scanned by the sensor usually only includes about 30 to 40 minutiae [7].

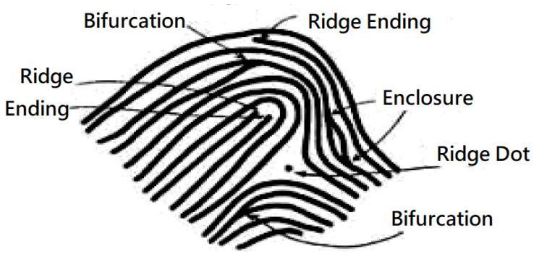


Fig. 1. Different ridge features on Fingerprint image [9]

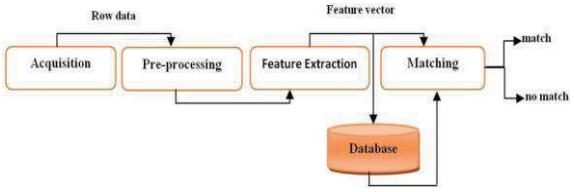


Fig. 2. The four stages of fingerprint recognition [9]

The main stages of fingerprint recognition include four process stages, image acquisition, image pre-processing, feature extraction, and matching. Fig. 2 shows the relationships among those four stages. The image acquisition stage is to obtain online or offline. Online fingerprint identification uses the optical fingerprint reader to capture the image of a fingerprint. Offline fingerprint identification puts ink on the fingertip, then puts a sheet of white paper on the fingerprint, finally scans the paper to get the fingerprint image. The pre-processing stage is to remove unwanted data, such as noise, reflection, .etc, in the fingerprint image to increase the clarity of ridge structure. Image segmentation, binarization, elimination of noise, smoothing, and thinning are the steps to enhance the fingerprint image. The feature extraction process of a fingerprint image is to locate, measure, and encode ridge endings and bifurcations in a fingerprint. Several methods are available to extract features from a fingerprint image; the famous one is the minutiae extraction algorithm. The matching stage calculates the degree of similarity between the user's input fingerprint image and an enrolled image from the database. Thus, this step is to compare the acquired feature with the template in the database. The three methods, the hierarchical approach, the classification approach, and the coding approaches, are available in this matching process [9].

2.2 LCG Pseudo-random number generator

Random numbers are a fundamental tool in many different areas; stochastic simulation and cryptography are the two main fields of applications [11]. In general, random numbers are critical in cryptography. We can use random numbers to encrypt e-mails, sign documents digitally, pay electronically, and so on. For cryptography, random numbers are unpredicted numbers. However, a computer cannot generate real random numbers; thus, we use the concept of pseudo-random numbers to produce a serial of random numbers in an unpredicted way. And, we use a pseudo-random number generator to generate a sequence of numbers in the interval [12]. A pseudo-random number generator (PRNG) is an algorithm to produce a serial of numbers that approximates the properties of random numbers. The sequence of numbers is not truly random; a random seed would determine it totally [13]. It is desirable to have the output of a PRNG practically indistinguishable and should not exhibit any correlations or patterns [14].

A well pseudo-random number generator should have two critical statistical properties, uniformity and independence [12]. Usually, good PRNGs should have several features, such as reproducibility and consistency (independent from the seed), portability, efficiency, and coverage of the entire output space. The design of PRNG should base on some practical aspects [15]. Also, a good PRNG should work efficiently, which means it can generate a large amount of random numbers in a short period; especially, for applications such as stochastic simulation, stream ciphers, the masking of protocols, or online gambling. Thus, it is necessary to use a fast PRNG when large

amounts of random numbers are required. Several suitable PRNG algorithms include linear congruential generator (LCG), lagged Fibonacci generator, linear feedback shift register [13], combined linear congruential generators (CLCG), random-number streams, and middle-square [12].

The LCG is one of the best-known PRNG algorithms; it is a simple linear congruential generator proposed by Lehmer. The LCG is a classical generator that uses a transition function [14] shown in Eq(1).

$$X_{n+1} = (a X_n + c) \bmod m \quad (1)$$

where a is the multiplier ($0 < a < m$), c is the increment ($0 < c < m$), and m is the modulus ($m > 0$), X_0 is the random number seed.

Although the processes of LCG PRNG are deterministic, it can show that the numbers generated by the sequence appear to be uniformly distributed and statistically independent with proper parameter settings. Therefore, the choice of a, c and m is critical. The LCG PRNG can pass the usual statistical tests only when it initializes its random seed properly and chooses the proper a, c, and m [12]. For generating total period random numbers, we should follow several principles to set the parameters, a, c, and m, in the LCG PRNG.

- m should be a prime and large, such as 231 – 1 for a 32-bit integer [12].
- It is good that m and c are relatively prime [13, 14].
- (a -1) is divisible by all prime factors of m [13].

Finally, Two LCGs can be combined to create a combined linear congruence generator, CLCG. With good parameter settings in the LCG PRNGs, the generator has a period that is the product of the period of each LCG PRNG [12].

2.3 AES encryption algorithm

AES is a symmetry block cipher developed by Joan Daemen and Vincent Rijmen; it is flexible to support any combination of data and secret key size of 128, 192, and 256 bits [16]. It uses two common techniques, substitution & permutation, to encrypt and decrypt data. AES encryption/decryption is an iterative process instead of Feistel cipher [17]. The fixed plaintext block size of AES is 128 bits (16 bytes). These 16 bytes construct a 4*4 matrix, named a state; AES encryption/decryption process operates on a matrix of bytes. Also, another critical feature in AES is the number of rounds, which depends on the size of a secret key. Three key sizes, 128, 192, or 256 bits, are available in AES; the key size determines the number of rounds in the AES encryption/decryption process. The AES has ten encryption/decryption rounds with a 128-bit key, AES has twelve encryption/decryption rounds with a 192-bit key, and AES has fourteen encryption/decryption rounds with a 256-bit key [17]. Fig. 3 shows the process flow of AES encryption/decryption process.

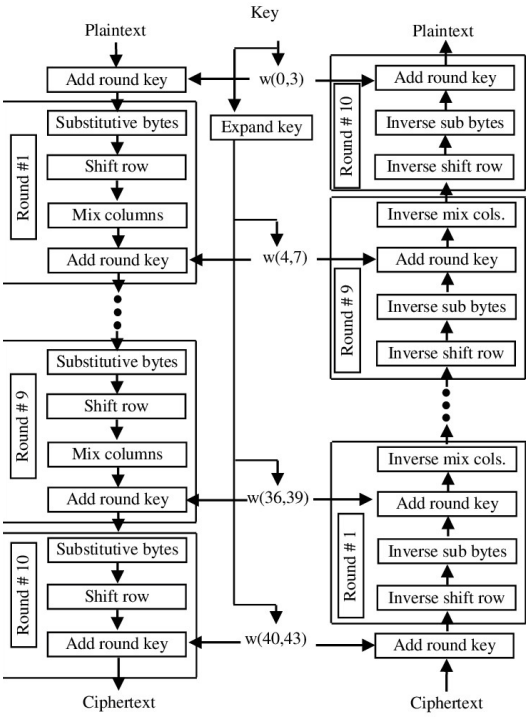


Fig. 3. The block diagram of AES encryption/ decryption process

AES uses four data transformation processes in a round to encrypt/decrypt data. The four sub-processes contain the 'Substitute Bytes Transformation', "the ShiftRows Transformation", the "MixColumns Transformation", and the "AddRoundKey Transformation" [17]. AES uses four data transformation processes in a round to encrypt/ decrypt data. The four sub-processes contain the "Substitute Bytes Transformation",

the "ShiftRows Transformation", the "MixColumns Transformation", and the "AddRoundKey Transformation".

The 'Substitute Bytes Transformation' is the first stage in one round. This stage uses a nonlinear substation table (S-box) to substitute a byte in the state for another byte. The AES depends on multiplicative inverse and affine transformation to construct S-box. The "ShiftRows Transformation" is the second stage in one round. This stage shifts bytes in each row to the left cyclically rather than row number zero. In this stage, row number zero does not perform any permutation. The first row only shifts one byte left circularly; the second row shifts two bytes left circularly; the last row shifts three bytes to the left circularly. The "MixColumns Transformation" is the third stage in the round. This stage performs a state multiplication process; each byte of one row in a set matrix multiplies by each byte of the state column; it uses a matrix multiplication of the state's columns to produce a new four bytes in the state. The "AddRoundKey Transformation" is the last stage in one round; it is the most vital stage for the AES to encrypt/decrypt data. This stage performs a simple XOR operation between the working state and the round keys; it can provide critical security for the AES during the data encryption process; and, this stage creates the relationship between the secret key and the ciphertext[16, 17].

III. A FINGERPRINT RECOGNITION-BASED ENCRYPTION MECHANISM

This study proposes a fingerprint recognition-based encryption mechanism to improve the confidentiality of the document files in a smartphone. The proposed encryption mechanism executes data file encryption /decryption operation based on a two-stage process over a smartphone; it is one of the identity-based cryptography [18, 19] mechanisms.

The following subsections will describe the proposed fingerprint recognition-based encryption mechanism in detail.

3.1 An overview of the fingerprint recognition -based encryption mechanism

The proposed fingerprint recognition-based encryption uses several techniques, such as fingerprint recognition, pseudo-random numbers, and AES encryption/decryption algorithm, to perform the two-stage encryption process. In the first stage, the proposed encryption mechanism depends on a fingerprint recognition function of a smartphone to identify whether the user has the right to access the specific document file in the smartphone or not. If the user passes the fingerprint recognition, he(she) has the right to perform an encryption/decryption process in the second stage.

The proposed encryption mechanism depends on generated random numbers and an AES algorithm to perform encryption/decryption processes in the second stage. Figure 4 shows the process flow of the proposed encryption mechanism. When users pass fingerprint recognition, the proposed encryption mechanism depends on the process option from the user's choice, encryption or decryption, to perform a different encryption/decryption process flow. In the encryption process, the proposed mechanism uses a pseudo-random number generator to generate a master key first; then, it encrypts the generated master key and saves the encrypted master key in the file header of an encrypted file; finally, it performs a file encryption operation with the AES algorithm. In the decryption process, the proposed mechanism decrypts the encrypted master key from the header of the encrypted file; then, it performs a file decryption operation with the AES algorithm. Fig. 5 shows the process flow in the proposed encryption mechanism.

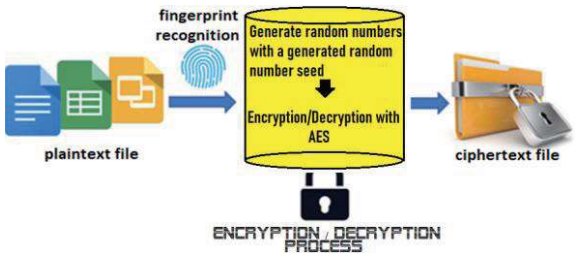


Fig. 4. A diagram of the process flow of the fingerprint recognition-based encryption mechanism

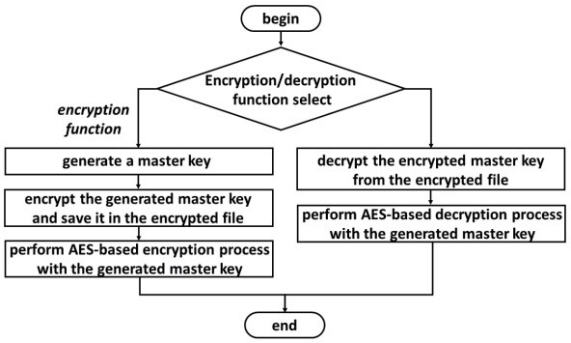


Fig. 5. The encryption/decryption process flows in the proposed encryption mechanism

Moreover, the proposed encryption mechanism will implement with Java to call fingerprint recognition function in an Android smartphone; however, it is easy to disassemble a Java-based app to realize an encryption/ decryption function in an app. Therefore, this study implements the core encryption function, a random number-based AES function, of the proposed encryption mechanism with C to enhance the security of the developed app. Therefore, this study uses the JNI (java native interface) to call the random number-based AES program with C in the Java developed app.

3.2 Dynamic secret keys generation

Since the proposed encryption mechanism depends on the AES algorithm to perform an encryption/decryption process, the AES algorithm is a symmetry encryption mechanism; it requires a secret key to perform the encryption/decryption operations. This study depends on the name and length of an encrypted file as a random number seed to generate a serial of random numbers. Those random numbers will be the master key in the proposed encryption mechanism and will become the secret keys for the AES algorithm to perform the encryption or decryption process in the proposed encryption mechanism. In general, different files can receive their corresponding random number seeds and a dynamic master key since there exist differences in their file names and lengths. Therefore, the proposed encryption mechanism depends on a file's name and size to produce a master key dynamically in the encryption process. The proposed encryption mechanism uses several techniques, such as ASCII code, bit shift, and congruence calculation, to calculate the random number seed. Fig. 6 shows the random number seed calculation based on all ASCII codes in a file name. For decrypting an encrypted file, the encryption process stores the length of a generated master key and a master key in the file header of an encrypted file. Fig. 7 shows the file header's content of an encrypted file.

Also, the proposed encryption mechanism calculates a master key length and generates a master key with the enhanced LCG [20]. Fig. 8 shows how the proposed mechanism calculates a master key length and generates a master key with the enhanced LCG. Moreover, Fig. 9 shows a diagram of the master key generation with random numbers generated by the enhanced LCG.

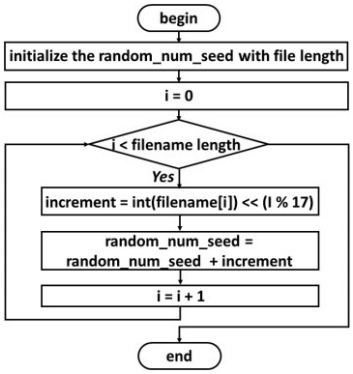


Fig. 6. The flowchart of the random number seed calculation in the proposed mechanism

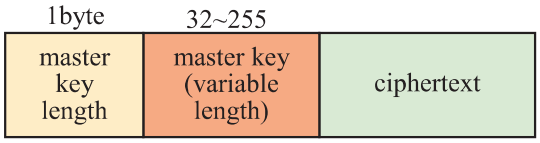


Fig. 7. The file header's format of an encrypted file

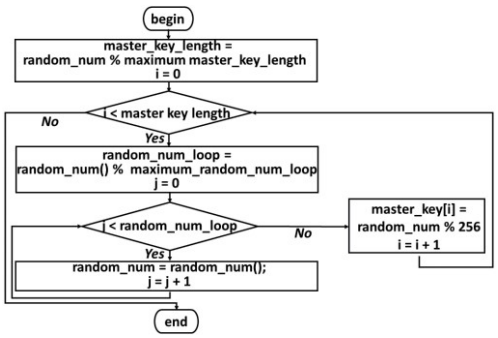
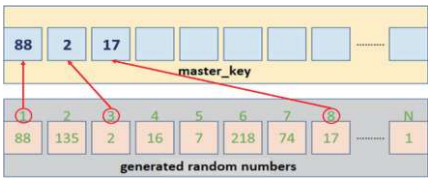


Fig. 8. The flowchart of a master key length and a master key generation



A serial of random numbers (1, 2, 5,….) uses to select the content of a master key

Fig. 9. A diagram the master key generation with random numbers

Since the proposed encryption mechanism stores a generated master key in the header of the encrypted file for decrypting the encrypted file, it is easy to be compromised by the opponent. Therefore, this study uses two measures to protect the generated master key. One is to encrypt the generated master key; the other is to transposition the encrypted master key. This study uses a lightweight stream cipher to encrypt the generated master key with a serial of generated random numbers. We depend on the file's name, length, and several techniques, such as bit shift and modulus operation, to produce a random number seed; then, the proposed encryption mechanism generates a serial of random numbers for the lightweight stream cipher. The values of the generated random number are 0 ~ 255. We base on the ASCII codes of the master key and the generated random number to have three scramble processes, XOR operation, ASCII code addition, and ASCII code subtraction. Fig. 10 shows the master key encryption process with the XOR, ASCII code addition, and ASCII code subtraction.

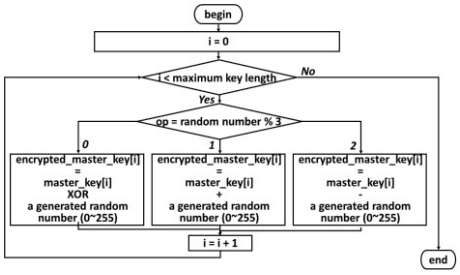


Fig. 10. The flowchart of the scramble process for the master key with three operations

Also, this study uses a transposition technique to store the encrypted master key in the header of the encrypted file randomly. The proposed encryption mechanism depends on the master key's length to generate a serial of random numbers; then, it gets lots of new positions, which store each byte of the encrypted master key in different locations. The scrambled master key can reduce the possibility of the encrypted file being compromised and enhance the security of the encrypted file. Fig. 11 shows a scrambled way that the proposed mechanism transpositions the encrypted master key with generated random numbers. Fig. 12 shows the file header of an encrypted file that contains the length of a master key and a scrambled encrypted master key.

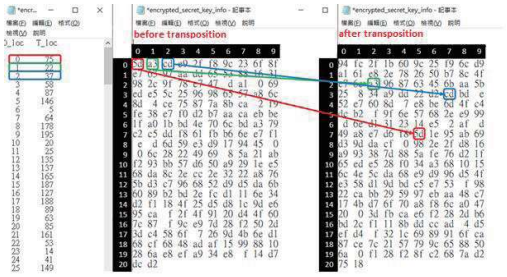


Fig. 11. A diagram of the transposition process of an encrypted master key

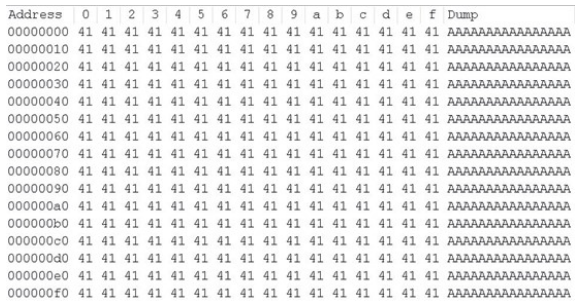
Fig. 12. The content of an encrypted file's header

3.3 AES-based encryption/decryption process

The proposed encryption mechanism performs a file encryption/decryption process with the AES algorithm in an Android smartphone; it uses 256 bits as a secret key for the AES to strengthen the robustness of the encrypted file. In general, there are many states in a file; the AES algorithm uses the same secret key, no matter with 128 bits, 192 bits, or 256 bits, to encrypt/decrypt all states in the file. However, the AES algorithm can encrypt the same plaintext to get the corresponding ciphertext (please see Fig. 13).

For avoiding the above situation, the proposed encryption mechanism uses a

generated dynamic master key as a set of secret keys; the AES algorithm encrypts states with different secret keys that can enhance the confidentiality of an encrypted file and reduce the compromised possibility of the encrypted file. This study uses a sliding window method to select a secret key required by the AES algorithm from the master key. Fig. 14 shows the diagram of secret keys selection from a generated master key with a sliding window method. The proposed encryption mechanism depends on the state process counter, the master key length, the secret key length (32 bytes), and a congruence operation to calculate the locations that each byte of a secret key should select from the master key. Fig. 15 shows how the proposed mechanism select a secret key from a generated master key.



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000010	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000020	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000030	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000040	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000050	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000060	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000070	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000080	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000090	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000a0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000b0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000c0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000d0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000e0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
000000f0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA

(a) plaintext

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000010	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000020	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000030	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000040	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000050	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000060	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000070	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000080	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
00000090	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000a0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000b0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000c0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000d0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000e0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#
000000f0	72	51	11	d0	31	62	c3	a2	dd	d3	ca	8b	8c	27	b9	23	rQ.?b??'?#

(b) ciphertext

Fig. 13. The AES algorithm comparison of plaintext /ciphertext with the same secret key

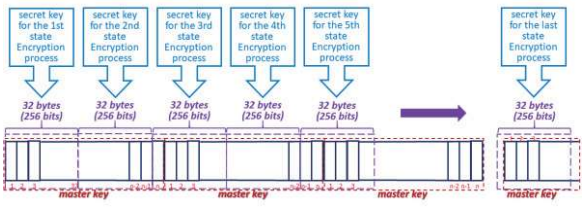


Fig. 14. A diagram of secret keys selection with a sliding window method

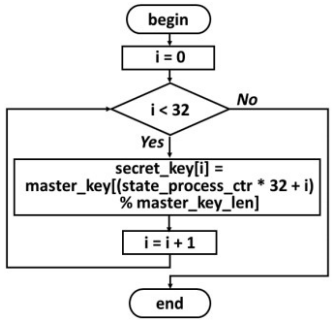


Fig. 15. The flowchart of a secret key selection

Since the lengths of all files cannot be divided by 16; therefore, in the encryption/decryption process, the proposed encryption mechanism depends on the size of a processed data block to determine which encryption algorithm it will use. If the processed data block is equal to the size of a state, that is 16 bytes; then, it uses the AES algorithm to perform the encryption/decryption process; otherwise, it uses a stream cipher algorithm to execute the encryption/decryption process with several generated random numbers. If the proposed mechanism performs an AES-based encryption process, it will select a secret key required by the AES algorithm from the master key first; then, it does the encryption process with the AES algorithm. If the proposed mechanism performs a stream cipher-based encryption process, it will generate several random numbers for the stream cipher algorithm first; then, it executes the encryption/decryption process with the stream cipher algorithm. Fig. 16 shows that the proposed encryption mechanism depends on the processed data length to use the AES algorithm or a stream cipher mechanism to perform encryption/decryption operations in the encryption/decryption process.

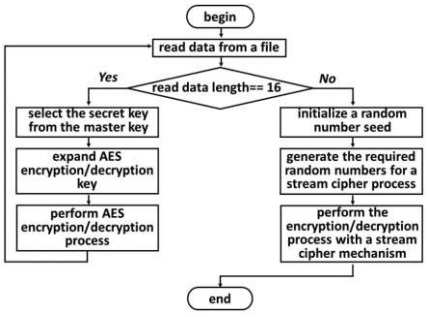


Fig. 16. The flowchart of the proposed encryption mechanism in the encryption/decryption process

IV. IMPLEMENTATION

In this section, we explain how to implement the proposed encryption mechanism in an Android smartphone first; then, we demonstrate the operation results of the proposed encryption mechanism in an Android smartphone.

4.1 Prototype program implementation

This study divides the prototype implementation of the proposed encryption mechanism into two parts. One is the user operation interface; the other is the AES-based encryption/decryption core program. Fig. 17 shows the process flow of the proposed encryption/decryption process.

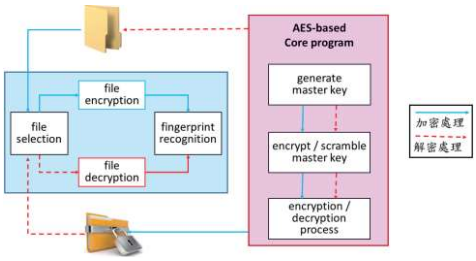


Fig. 17. A diagram of the encryption/decryption process flow in the proposed mechanism

Since the proposed encryption mechanism would operate in an Android smartphone, this study implements the user operation interface of the proposed encryption mechanism with the Android Studio SDK (Software Development Kit) under the Windows 10 platform. We implement an encryption app with Java codes in the Android Studio SDK as the user operation interface of the proposed encryption mechanism. In the developed encryption app, we code Java to import the BiometricPrompt package supported by an Android platform with an Android API (Application Programming Interface). Also, we call an AES-based core program coded by C with the JNI in the developed app.

This study depends on the proposed encryption mechanism mentioned in Section 3 to code the AES-based encryption/decryption core program with standard C. For called by the developed encryption app, we follow the JNI calling format to code the argument's interface in the AES-based encryption/decryption core program. Then, the developed encryption app can call the AES-based encryption/decryption core program with the JNI to perform an encryption/decryption process for a selected file.

4.2 Encryption/decryption operation with the developed encryption app

In the operation of the developed encryption app, the user should select a file that he(she) wants to encrypt/decrypt first; then, he(she) needs to choose an operation function, encryption or decryption (please see Fig. 18). When he(she) finishes a file selection, the developed encryption app shows the file path of the selected file (please see Fig. 19). If the user choices the encryption operation, the developed app performs the file encryption process. The file encryption process executes the fingerprint recognition function first. If the user passes the fingerprint identification (please see Fig. 20), the encryption process generates a ciphertext file with an extension filename, enc. If the user selects the decryption process, the file decryption process also executes the fingerprint recognition function first. If the user passes the fingerprint identification, the decryption process generates a recovery file from a ciphertext file with an extension filename, R. Fig. 21 shows the scrambled content in an encrypted file.

Fig. 18. The file and process selection in the developed encryption app

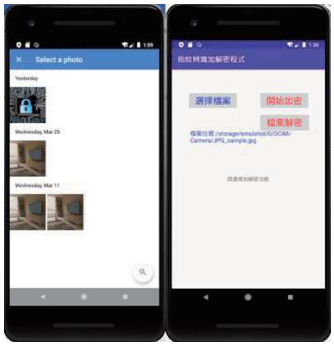


Fig. 19. The screens of file selection and file path of the selected file display

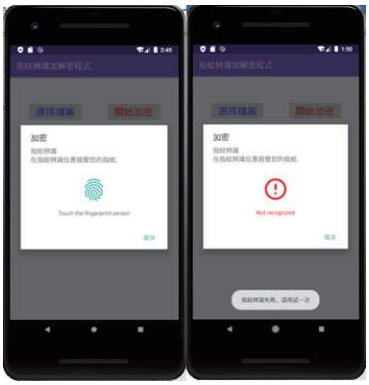


Fig. 20. Screens of fingerprint recognition function

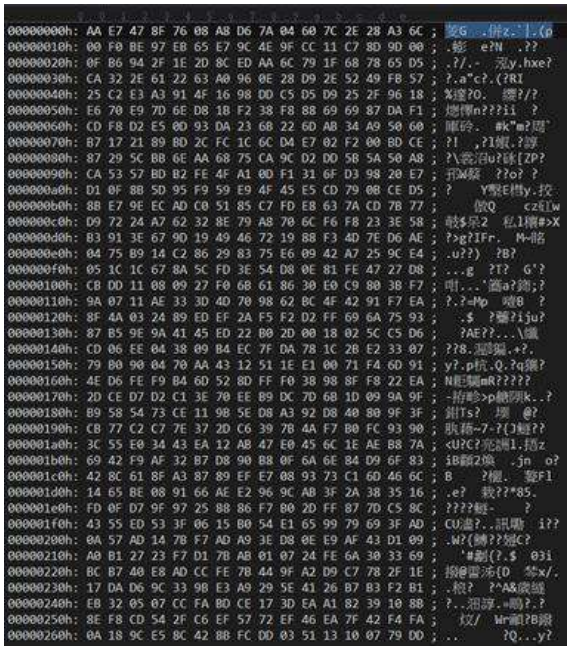


Fig. 21. The unreadable content of an encrypted file

4.3 A encryption/decryption correctness verification of the proposed encryption mechanism

For understanding the encryption/ decryption process correction of the proposed encryption mechanism, this study uses four types of frequently used files, such as doc, jpg, pdf, and ppt, to execute encryption and decryption processes with the proposed encryption mechanism. The encryption/ decryption results help us to understand whether the proposed encryption mechanism can handle an encryption/decryption job for different formats of files or not. Moreover, the correct content of a recovery file that decrypts from an encrypted file is the only criterion to verify whether an encryption/decryption mechanism can correctly perform an encryption/decryption process or not. Therefore, this study adopts two ways to compare the file contents of a plaintext file and a recovery file. One way is to use the "comp" command in the Windows command prompt operation environment to compare the contents of two files; the other way depends on the hashing codes of two files to identify the contents of those two files are the same or not. This study uses several well-known hashing functions to produce a hashing code of a file.

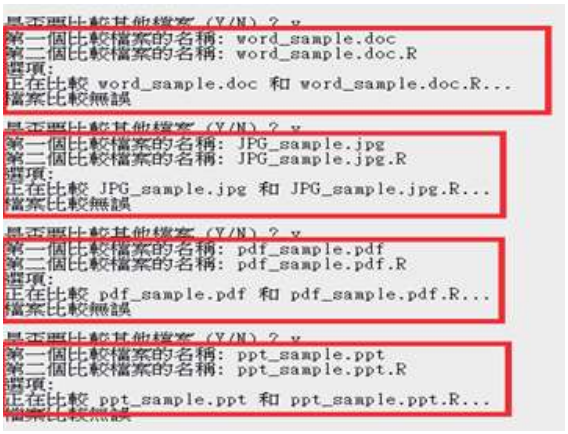


Fig. 22. The comparisons of the plaintext file and the ciphertext file with the “comp” command

Fig. 22 shows the comparisons of the four types of files with the "comp" command in the Windows command prompt operation environment (Since we use the traditional Chinese version of Windows operating system, therefore, Chinese information appears in the comparison result of comp command.). Looking at Fig 22, the four types of files receive the same content in their plaintext file and recovery. Also, we use MD5, SHA-1, and SHA-256 to compare the plaintext files and the recovery files of those four types of files (please see Table 1). Table 1 shows that the plaintext file and the recovery file of each file format receive the same hashing codes in MD5, SHA-1, and SHA-256. According to Fig. 22 and Table 1, We find that the proposed encryption mechanism can correctly recover plaintext files from encrypted files. This result demonstrates that the proposed encryption mechanism can perform an encryption/decryption process for different types of files well.

Table 1. The hashing codes of the four types of plaintexts file and recovery files

MD5	DOC	9d67d4e76d11f0616abed46f4e051798
	DOCR	9d67d4e76d11f0616abed46f4e051798
	JPG	b1a073c551315b31d6daa75203eb0e43
	JPGR	b1a073c551315b31d6daa75203eb0e43
	PDF	73ed59f5294938dc6ff038536cfe3ecb
	PDFR	73ed59f5294938dc6ff038536cfe3ecb
	PPT	ddd28070f16c5f22dc4267a78b4cc368
	PPTR	ddd28070f16c5f22dc4267a78b4cc368
SHA1	DOC	ab34f7f8f0184d8a0ce6220e96719df325cc8804
	DOCR	ab34f7f8f0184d8a0ce6220e96719df325cc8804
	JPG	4d8351d1df3f904a8fc251a6c0c07997203d7efa
	JPGR	4d8351d1df3f904a8fc251a6c0c07997203d7efa
	PDF	144876a441335f8f26f81d85289b84a999b75f48
	PDFR	144876a441335f8f26f81d85289b84a999b75f48
	PPT	4ac24cfa39881169bf94b42bf7539fc828c8e21f
	PPTR	4ac24cfa39881169bf94b42bf7539fc828c8e21f
SHA256	DOC	f812535c2beb050df7466d1162912e754bde8b1583c8bc4dc957ade585dbfb89
	DOCR	f812535c2beb050df7466d1162912e754bde8b1583c8bc4dc957ade585dbfb89
	JPG	a8f09f5dad6a1875ba1aca1257fbef9d5a55252ef6dd87b79159d553995707d7
	JPGR	a8f09f5dad6a1875ba1aca1257fbef9d5a55252ef6dd87b79159d553995707d7
	PDF	bf1059d5ab0906d20689efa4434f929a8531b84fce8444d4d676bcdad7cf60fe
	PDFR	bf1059d5ab0906d20689efa4434f929a8531b84fce8444d4d676bcdad7cf60fe
	PPT	98a67863358524cb208cf6250492f562c717170d492903adb52463c4c91092bc
	PPTR	98a67863358524cb208cf6250492f562c717170d492903adb52463c4c91092bc

Legend :
DOC: word_sample.doc, DOCR: word_sample.doc.R,
JPG: JPG_sample.jpg, JPGR: JPG_sample.jpg.R,
PDF: pdf_sample.pdf, PDFR: pdf_sample.pdf.R,
PPT: ppt_sample.ppt, PPTR: ppt_sample.ppt.R

V. CONCLUSION

Smartphones have become useful mobile devices in our daily life; we always use them to store critical files. Therefore, it is a significant issue for users to improve the security of their smartphone files. Usually, data encryption is the best way to improve the confidentiality of vital files. Therefore, this study proposes a two-stage encryption/decryption mechanism to enhance the confidentiality of the files stored in an Android smartphone with fingerprint recognition measures. The first stage in the proposed encryption mechanism uses the fingerprint recognition function supported by an Android smartphone platform to identify the user who wants to access a critical file in the Android smartphone. If the user passes the fingerprint identification, the proposed encryption mechanism performs an encryption/decryption process with the AES algorithm in the second stage; otherwise, the proposed encryption mechanism will refuse to access a critical file in an Android smartphone.

For understanding the feasibility and correction of the proposed encryption mechanism in an Android smartphone, this study implements the proposed encryption mechanism with Java and C. This study also uses several scenarios to verify the correction of the encryption/decryption process supported by the proposed encryption mechanism. The results show that the proposed encryption mechanism can correctly encrypt and decrypt different files in an Android smartphone with a fingerprint recognition measure. Thus, the user can improve the confidentiality of his/her critical files stored in an Android smartphone with the proposed encryption mechanism. The study result demonstrates that the proposed encryption mechanism can enhance the security of important files with fingerprint recognition in an Android smartphone.

The proposed mechanism supports encryption/decryption operation on a smart mobile device based on fingerprint recognition; therefore, it has a limitation that a smartphone must equip with a fingerprint recognition device. However, this limitation will be disappearing gradually. A smartphone with a fingerprint recognition device and cameras would be a trend. Thus, a smartphone can support fingerprint recognition and face identification functions easily. Also, for improving the security of files on a smartphone with functionality supported by a smartphone, encrypting confidential files with face identification would be one possible research issue in the future.

REFERENCES

[1] Colwill, Carl. "Human factors in

information security: The insider threat-Who can you trust these days?." Information security technical report 14.4, 186-196, 2009.

[2] Fu, Chen-Hua and Chen, Chih-Yung. "A Study on Decision-Making Opinion Exploration in Windows-Based Information Security Monitoring Tool Development." Sustainability 13.7: 3815, 2021.

[3] Alotaibi, Sara Jeza, and David Argles. "FingerID: A new security model based on fingerprint recognition for personal learning environments (PLEs)." 2011 IEEE Global Engineering Education Conference (EDUCON). IEEE, 2011.

[4] Xiao, Qinghan. "Security issues in biometric authentication." Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. IEEE, 2005.

[5] Yang, Wencheng, et al. "Security and accuracy of fingerprint-based biometrics: A review." Symmetry 11.2 (2019): 141.

[6] Gordon, Rob. Essential JNI: Java Native Interface. Prentice-Hall, Inc., Hoboken, New Jersey, 1998.

[7] Ton, Van der Putte and Keuning, Jeroen. "Biometrical fingerprint recognition: don't get your fingers burned." Smart Card Research and Advanced Applications. Springer, Boston, MA, 2000. 289-303.

[8] Zhuang, Zheng-Yun, et al. "A hybrid session key exchange algorithm for highly-sensitive IP-based institutional communications." Microsystem Technologies 24.1 (2018): 273-283.

[9] Ali, Mouad MH, et al. "Overview of fingerprint recognition system." 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT). IEEE, 2016.

[10] Datta, A. K., Lee, H. C., Ramotowski, R., & Gaensslen, R. E.. Advances in fingerprint technology. CRC press, Boca Raton, Florida, 2001.

[11] Vajargah, Behrouz Fathi, and Rahim Asghari. "A novel pseudo-random number generator for cryptographic applications." Indian Journal of Science and Technology 9.6 (2016): 1-5.

[12] Vajargah, Behrouz Fathi, and Rahim Asghari. "A pseudo random number generator based on chaotic henon map (CHCG)." International Journal of Mechatronics, Electrical and Computer Technology (IJMEC) 5.15 (2015): 2120-2129.

[13] Mishra, Mina, and V. H. Mankar. "Text encryption algorithms based on pseudo random number generator." International Journal of Computer Applications 111.2 (2015).

[14] Mohanty, Siddhant, A. K. Mohanty, and F. Carminati. "Efficient pseudo-random number generation for monte-carlo simulations using graphic processors." Journal of Physics: Conference Series. Vol. 368. No. 1. IOP Publishing, 2012.

[15] Datcu, Octaviana, Corina Macovei, and Radu Hobincu. "Chaos based cryptographic pseudo-random number generator template with dynamic state change." Applied Sciences 10.2 (2020): 451.

[16] Zeghid, Medien, et al. "A modified AES based algorithm for image encryption." International Journal of Computer Science and Engineering 1.1 (2007): 70-75.

[17] Abdullah, AkoMuhamad. "Advanced encryption standard (AES) algorithm to encrypt and decrypt data." Cryptography and Network Security 16 (2017): 1-11.

[18] Wu, Chien-Hsing, Jing-Jang Hwang, and Zheng-Yun Zhuang. "A trusted and efficient cloud computing service with personal health record." 2013 International Conference on Information Science and Applications (ICISA). IEEE, 2013.

[19] Narayana, V. Lakshman, and C. R. Bharathi. "Identity based cryptography for mobile ad hoc networks." Journal of Theoretical and Applied Information Technology 95.5 (2017): 1173.

[20] Deng, Ming-Yan, "A Study on Symmetric Encryption Scheme with Enhanced Pseudo Random Number and Streaming Cipher Skill." master thesis, Department of Information Management, College of Management, National Defense University, 2009. <https://hdl.handle.net/11296/w7s332> (Visited time : Oct. 15, 2021)