中正嶺學報 第五十三卷 第一期 民國 113. 05
JOURNAL OF C.C.I.T., VOL. 53, NO. 1, MAY, 2024
DOI：10.30188/JCCIT.202405_53(1).0006

A Study on AES-based Encryption Scheme with Face Recognition Mechanism

Chen-Hua Fu[1*], Chung-Wen Tzeng[2], Hsin-Wei Kuo[2], Jin-Sheng Chu[1]

[1] *Department of Information Management, College of Management, National Defense University*
[2] *Department of Industrial Engineering and Systems Management, Feng Chia University*

ABSTRACT

Since more smartphones and notebook computers are available and face-recognition algorithms are mature, this study proposes an AES-based encryption scheme with a face-recognition mechanism to solve an issue of key management in a symmetric encryption mechanism. The proposed encryption scheme depends on the content of a plaintext file to generate a secret key with a hashing function; then, it encrypts the plaintext file; finally, it stores all information required in the decryption process into a ciphertext file. And; the proposed encryption scheme depends on the face-recognition result to decide whether one person in front of a camera can decipher the ciphertext file. This study implements the proposed encryption scheme with Python and executes several scenarios. The execution results confirm the feasibility and practicality of the proposed encryption scheme; they also demonstrate that the proposed encryption scheme can solve the secret key management issue in a symmetric encryption mechanism.

Keywords: AES encryption mechanism, face-recognition, hashing function, secret key management

應用人臉辨識機制於以 AES 為基加密機制之研究

傅振華[1,*] 曾中文[2] 郭修瑋[2] 初璟珅[1]

[1] 國防大學管理學院資訊管理學系
[2] 逢甲大學工業工程與系統管理學系

摘　　要

隨著智慧型手機和筆記本電腦的普及以及人臉辨識演算法的成熟，本研究提出一種應用人臉辨識機制於以 AES 機制為基的加密方案，以解決對稱式加密機制所衍生的密鑰管理問題。研究所提出的加密方案將依據於明文文件的內容及雜湊函數產生文件加解密作業所需的密鑰；然後，進行明文文件加密處理；最後，它將解密過程中所需的所有資訊存儲到密文文件中。同時;研究所提出的加密方案將取決於人臉辨識的結果來決定鏡頭前的人是否具有解密密文文件的權利。本研究透過 Python 程式語言撰寫研究所提出加密方案的雛型程式並執行多個想定。執行結果驗證了所提出的加密方案的可行性和實用性；它們還證明研究所提出的加密方案可以解決對稱加密機制中的密鑰管理問題。

關鍵詞：AES加密機制、人臉辨識、雜湊函數、密鑰管理

文稿收件日期 112.7.04; 文稿修正後接受日期 112.11.22; *通訊作者
Manuscript received July 04, 2023; revised Nov 22, 2023; * Corresponding author

I. INTRODUCTION

It highlights the importance of information security with the evolution of information and network technology; the confidentiality of documentation becomes a critical information security issue. People always use encryption schemes to ensure the confidentiality of considerable documentation. However, an encryption scheme requires a secret key or a pair of public keys; usually, more documents encrypt, and more secret keys / public keys need. It might be a problem for users to remember those keys used in documentation encryption. The key management of documentation encryption would be an issue that require us to explore.

Due to the maturity of information and communication technology, many mobile devices, such as smartphones and notebook computers, appear in daily life and work. Usually, those smartphones and notebook computers have an embedded camera device; therefore, people can have a camera device available when they can quickly and easily access smartphones and notebook computers. In addition, face recognition algorithms are becoming more and more mature. We can adopt a face recognition scheme with those available cameras to strengthen the documentation encryption process.

Since a secret key management issue exists and face-recognition algorithms mature, this study proposes an AES-based encryption scheme with face-recognition mechanism. The proposed encryption scheme uses a hashing function to generate a secret key for a cipher job and avoids the secret key management issue in an encryption scheme; it also stores all information required in the decryption process into a ciphertext file. Moreover, the proposed encryption scheme depends on the face-recognition result to decide whether the person in front of the camera can perform a decryption process; the face-recognition result would be a threshold to trigger the execution of a decryption process in the proposed encryption scheme.

We organize the remainder of this paper as follows. Section II discusses the related technologies adopted in the proposed encryption scheme. We have detailed descriptions of the proposed encryption scheme in Section III. Section IV presents the prototype program implementation of the proposed encryption scheme and the execution results of some scenarios with the prototype program. Finally, we have a conclusion in Section V.

II. RELATED WORKS

This study adopts several technologies, face recognition, SHA hashing algorithm, and AES-GCM mechanism, to construct the proposed encryption scheme; we will have literature reviews on those technologies.

2.1 Face recognition

In general, biometric identification depends on physiological features, such as fingerprint, iris pattern, or face, to identify a person. And; as smartphones prevail, face recognition becomes one of the primary biometric technologies; it adopts automated methods to recognize the identification of a person with his facial features [1]. Compared with other biometric technologies, face recognition has several advantages, such as natural, nonintrusive, and ease of use [2].

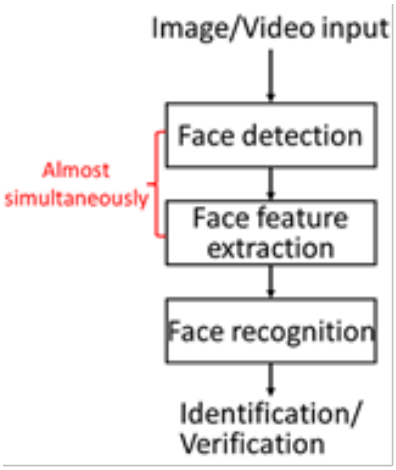

Fig. 1. A procedure of a computer-based face recognition (Source: [3])

Usually, computer-based face recognition involves three steps: Step 1 face detection from cluttered scenes; Step 2 feature extraction from the detected face regions; Step 3 identification/verification (See Figure 1) [3]. The execution order of those three steps is sequential. And, the face detection and the facial feature extraction perform almost simultaneously.

Face detection will separate a face region from a background image in a picture. Hjemal and Low use two categories of face-detection techniques; one is feature-based techniques, and the other is image-based techniques. The

DOI：10.30188/JCCIT.202405_53(1).0006

feature-based methods use a person's facial features in their detection process; the mechanisms used in the feature-based approaches include low-level analysis, feature analysis, and active shape model [4]. The image-based methods usually depend on new study fields, such as machine learning; the mechanisms adopted in the image-based approaches are neural networks, example-based learning, and support vector machine [5, 6]. When a face recognition mechanism detects a face region in an image, it will extract a facial features from the detected face region at the same time. The facial features are a set of structural encodings to describe the presented face in the detected area. Those feature encodings include center-view descriptions and some abstract descriptions of the global configuration and features [7].

The face recognition mechanisms depend on the three main approaches, Holistic, feature-based, and hybrid, to complete face recognition jobs [3]. The Holistic Approach takes a whole face region as input data with various methods, such as eigenfaces, fisher faces, support vector machine, and hidden Markov model (HMM), which are the principal component analysis-based (PCA-based). The feature-based approaches analyze local features of facial organs, such as the nose, eyes, mouth, and their geometric relationships; this approach also calls the geometric feature-based approach [8]. Since human facial features play a critical role in face recognition, many researchers have determined that the eyes, mouth, and nose are the most significant features. Therefore, the hybrid approach will combine local features and all features of a detected face; usually, it will use a modular eigenface with local feature methods [9]. Also, several approaches, such as the geometry-based technique, template-based technique, appearance-based approach, and color-based method, have been used to extract the facial features from the detected images [10].

2.2 SHA hashing algorithm (SHA)

A hash algorithm is a one-way function; usually, it takes an arbitrary length message to produce a fixed-size message digest string [11]. Essentially, a message has its corresponding unique message digest. Therefore, we can use hash functions to check the integrity of one specific message. Also, hash functions can provide authentication services with digital signature and message authentication code (MAC) algorithms [12].

Secure hash algorithm-1 (SHA-1) is the world's most popular hash function used in cryptography; it takes a limited-size message as an input and produces a 160-bit message digest (hashing value)[13]. However, SHA-1 security strength is comparable to 80-bit block encryption, but its security strength is somewhat limited [14]. There might exist a security concern with SHA-1.

SHA-2/SHA-3 are new standards in the hash function families; it has many changes from SHA-1. SHA2 is composed of six hash functions to generate message digests in different lengths; the length of those digests is 224, 256, 384, or 512 bits [13]. SHA-1 and SHA-2 adopt the same functional structure, but they use some variations in the internal operations, such as message size, message block size, word size, number of security bits, and message hash size [15]. SHA-3 is based on the KECCAK algorithm; its family consists of four cryptographic hash functions, named SHA3-224, SHA3-256, SHA3-384, and SHA3-512 [16]. Table 1 shows those variations.

Table 1. Secure Hash Algorithms

Algorithm	SHA-1	SHA-256	SHA-512	SHA3-256	SHA3-512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	any size	any size
Block size	512	512	1024	136	72
Message digest size	160	256	512	256	512

2.3 AES-GCM

The AES (Advanced Encryption Standard) [17] is one of the symmetric block encryption algorithms; US NIST (National Institute of Standards and Technology) selected it as a federal information processing standard - FIPS PUB 197 [18]. The AES algorithm uses a secret key of 128/192/256 bits to encrypt/decrypt a 16-byte (128 bits) block [17].

The AES depends on content

substitution/location transportation operations to perform a cipher/decipher job with 10, 12, or 14 rounds of encryption/decryption. It bases on the secret key length to determine the exact round number of encryption/decryption. Usually, a typical round of AES algorithm includes four types of operations, Substitution Bytes, Shift Rows, Mix Columns, and Add Round Key. And; the final round omits the Mix Columns operation; it is slightly different from a normal round process [19]. Usually, people use the AES to execute an encryption process with a longer secret key; they can receive more security robustness in the ciphertext to resist the brute-force attack of a secret key. The AES-256 mechanism with the 14 encryption/ /decryption rounds can resist a key-recovery attack with a total complexity of 2^{131} time and 2^{65} memory [20, 21].

The AES-Galois/Counter Mode (GCM) is one of the AES-based block encryption algorithms [22]. It is a critical encryption scheme in cryptographic practice; for example, IPsec and TLS adopt it [23]. The AES-GCM requires four inputs, an AES-based secret key, an initialization vector (it is a nonce), plaintext, and additional authenticated data (AAD) that is optional. Moreover, it produces two outputs, a ciphertext and a message authentication code (an authentication tag).

The IV is a nonce essentially; its value is unique for a specified context; it determines an invocation of the authenticated encryption function on the protected plaintext. Moreover, the IV/nonce should be multiples of 8 so the IV/nonce can be byte strings; usually, the AES-GCM performs efficiently with the IV/nonce in 12 octets, a 12-octet IV/nonce is recommended [24].

2.4 Information security schemes with face recognition

A facial feature is one of the biometrics features; it is often integrated with encryption schemes to enhance the robustness of information security. Using a face recognition mechanism to strengthen related information security schemes has become a trend. Venkatesan, R., et al. proposed a two-way authentication system with facial and proxy detection to improve security during online transactions. The system uses 128 facial feature points of the user to increase the accuracy of the system and a triple-DES encryption mechanism to enhance the security of the work [25]. Ameen introduced a secure e-e web application with a secured authentication process. The secured authentication process depends on the single elector's selection of encryption and face recognition [26]. Battaglia, F., et al. A multi-factor authentication system is proposed based on a dual cascade classifier for face recognition and encrypted RFID tags for token-based authentication. They avoid using a centralized facial recognition database and store these sensitive facial recognition data in RFID; furthermore, this approach makes the system performance independent of the total number of registered subjects [27]. Sawant, V. A., et al. combined a facial recognition technology with the AES encryption algorithm to propose a novel approach to enhance the security of password-based encryption systems. They used a facial recognition mechanism as an extra authentication process to verify the user's identity before allowing access to encrypted data [28]. Vankadara, A. et al., proposed a new security scheme. The scheme increases the data security of a database with SHA-512 encryption and adopts a second level authentication with machine-learning-based face recognition [29]. Chandrasekhar et al. proposed a new homomorphic encryption scheme with facial templates to guarantee the security of the cloud user information and the person's reclusive identification; the proposed encryption scheme performs a double abstraction methodology to assure data protection on a cloud computing platform with facial templates. The usage of Facial templates is a critical aspect in the encryption process [30].

III. AN AES-BASED ENCRYPTION SCHEME WITH A FACE RECOGNITION MECHANISM

Face recognition mechanisms depend on extracted face feature vectors to verify the identification of a person. A person would get his corresponding face feature vectors from a camera with a face recognition mechanism. Usually, two people cannot get pretty close face feature vectors with a face recognition mechanism from a

DOI：10.30188/JCCIT.202405_53(1).0006

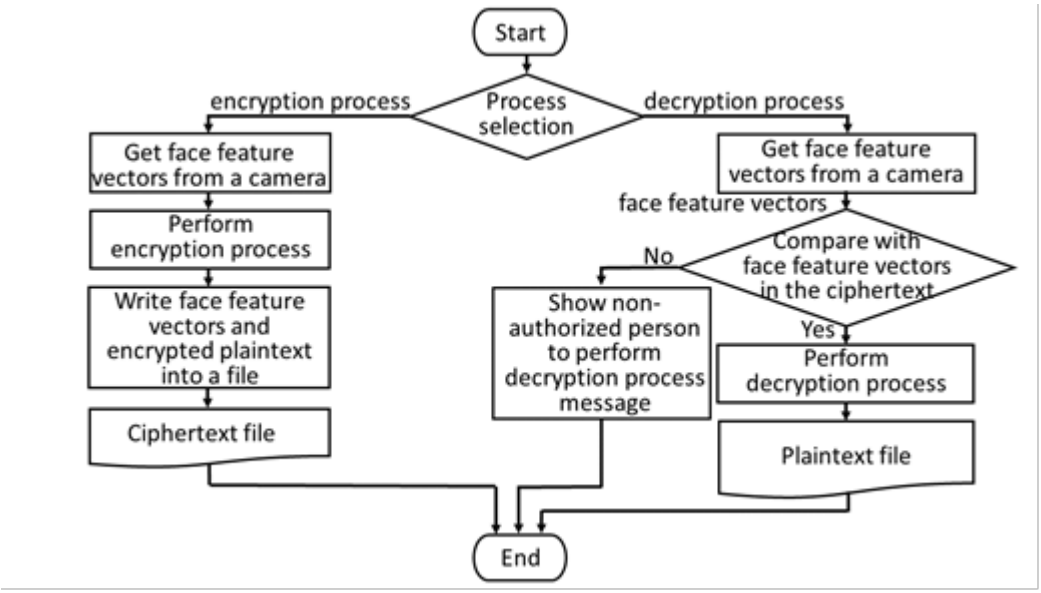

Fig. 2. A flowchart of encryption process / decryption process in the AEFR scheme

camera. Therefore, this study will use the face feature vectors of a person in an AES-based encryption scheme to verify a person's authorization to perform a decryption process of a ciphertext file. Therefore, this study proposes an AES-based encryption scheme with a face recognition mechanism, named the AEFR scheme, to enhance the confidentiality of documentation in intelligent devices with a camera. The AEFR scheme adopts a face recognition mechanism in both the proposed encryption process and decryption process; it will use a set of face recognition feature vectors get in the encryption process to verify whether a person who is authorized person to decrypt a ciphertext file in the decryption process or not.

Figure 2 shows that the AEFR scheme depends on users' selection to perform an encryption or decryption process. First, both encryption and decryption processes will get a set of face feature vectors from a camera; then, the AEFR scheme will perform the subsequent encryption/decryption process. The encryption process will depend on an AES-GCM algorithm to encrypt a plaintext file; then, it writes the extracted face feature vectors and the encrypted result into a ciphertext file. The decryption process will get the face feature vectors of a person from a camera and have a comparison with the face feature vectors in the ciphertext file first. If the person has the authority to decrypt the ciphertext file, the decryption process deciphers the ciphertext file and writes deciphered plaintext results into a plaintext file; otherwise, the decryption process will display a message that shows the person in front of the camera is an unauthorized person to perform the decryption process of the ciphertext file.

3.1 Face feature vector extraction

A set of facial features is one of the critical biological features; we can use it to verify the identification of a person. The AEFR scheme will depend on the face feature vectors stored in a ciphertext file and extracted from a camera to determine whether a person can decrypt the ciphertext file or not. Therefore, it is a critical process for the AEFR scheme to extract face feature vectors from a camera.

For extracting a set of face feature vectors of a person from a camera, the AEFR scheme will open a camera first; then, it captures an image that contains a person's face from the opened camera. Next, the AEFR scheme uses a DLIB-based face recognition module to find the face location of a person in the captured image

and extracts a set of face feature vectors from the found face location. Finally, the AEFR scheme keeps the extracted face feature vectors for sequential encryption/decryption. Figure 3 shows the flowchart of a face feature vector extraction.

Fig. 3. A flowchart of a face feature vector extraction in the AEFR scheme

3.2 Encryption process

The encryption process of the AEFR scheme is an AES-GCM-based encryption mechanism with multiple variable blocks. For solidating the confidentiality of a ciphertext file, the encryption process divides a plaintext file into many variable-size blocks and encrypts those variable-size plaintext blocks with the AES-GCM mechanism. And it uses a random number to decide an encryption block size in each round of encryption. Moreover, to let users could only depend on their faces to perform an encryption process without inputting extra information, the AEFR scheme uses several functions to archive this goal. Those functions include a face recognition function, a master-key generation function, a random number seed generation function, a secret-key selection function, and a data scramble function. Those functions depend on applications of a hashing operation, a pseudo-random number generation, and an XOR operation.

- A face recognition function

Since it is impossible for a face feature extraction algorithm to get the same face feature vectors in every face recognition process, the AEFR scheme cannot use extracted face feature vectors as a secret key in a symmetric encryption mechanism. Due to this reason, the AEFR scheme uses extracted face feature vectors from a person as a threshold for determining whether a person can do a decryption job. Therefore, the encryption process of the AEFR scheme will capture an image of a person who performs a documentation encryption process from a camera; then, it extracts a set of face feature vectors from the captured image and stores them in the ciphertext file.

- A master-key generation function

In general, an SHA hashing operation depends on the content of a message to generate its corresponding unique message digest; only the same message content can produce the same message digest. The AEFR scheme generates a master key for the encryption process with the message digest of the plaintext file. Thus, only the person who gets the correct contents of the plaintext file can get the right master key in an encryption/decryption process. For increasing the change of secret key selection from the generated master key in an encryption process, the AEFR scheme uses SHA512 hashing operation to produce a 64 bytes message digest of a plaintext file as a master key for the proposed encryption to do encryption/decryption jobs with the AES-GCM mechanism. With a 64-byte message digest, there are 256 power of 64 combinations for a master key.

- A random number seed generation function

For getting a series of controllable random numbers in the encryption/decryption process, the AEFR scheme uses the 64-byte master key to generate a random number seed; it uses the ASCII code of the 64-byte master key and the location of a character in the 64-byte master key to calculate the random number seed with simple addition and multiply arithmetic operations. Figure 4 shows a pseudo-code of the random number seed generation with the 64-byte master key. The encryption process depends on the generated random number seed to get a series of controllable random numbers to scramble the critical data, the 64-byte master key, face feature vectors, and pairs of nonces and tags, in the encryption process and the decryption process.

DOI：10.30188/JCCIT.202405_53(1).0006

```
random_num_seed = 0
for (l = 0; l < length of master key; I = l + 1)
  {
  j = l mod 6
  random_num_seed = random_num_seed
    +    ASCII    code    of    masterkey[l]     *j
}
```

Fig. 4. A pseudo-code of the random number seed generation in the AEFR scheme

- A secret-key selection function

The AES-GCM mechanism performs an encryption/decryption process with 256 bits(32 bytes) secret key; therefore, the AEFR scheme selects a required secret key from the master key randomly with 32 generated random numbers for the AES-GCM mechanism for each encryption/decryption process. The secret-key selection function depends on the master key size to generate a random number, which will be in the interval (0 ~ 63); then, it uses the generated random number as a location index of the master key. The secret-key selection function depends on the location index to retrieve a one-byte content in the master key; then, it assigns the one-byte content as a part of the selected secret key. Since the secret-key selection function might choose the contents of a required secret key from the identical location of the master key repeatly, the combination of a secret key choosing from the master key would be 64 power of 32. Figure 5 shows the pseudo code of the secret-key selection function.

```
for (i = 0; i < 32; i = I + 1)
  {
   loc = random number mod 64
     secret_key[i]                    =
master_key[loc]
```

Fig. 5. A pseudo-code of the secret-key selection function

- A data-scramble function

For storing the master key, face feature vectors, and pairs of nonce and tag in an unreadable pattern, the encryption process scrambles those data with two operations. One is a content substitution operation; the other is a location transposition operation. First, the data-scramble function performs the content substitution operation that depends on the ASCII codes of those data and a series of random numbers to scramble them with three simple arithmetic/logical operations, addition, subtraction, and XOR; it determines one of the three operations with the location index of one character in the processed data and substitutes the content of the specific character in an unreadable pattern. Figure 6 shows the pseudo-codes of the content substitution operation.

```
for (l =0; l < length of the processed data; I =
l+1)
  {
   If l mod 3 == 0:
   the ASCII code of scrambled_data[l] = (the
      ASCII code of the processed_data[l] +
      random_number) mod 256
   If l mod 3 == 1:
   the ASCII code of scrambled_data[l] = (the
      ASCII code of the processed_data[l] -
      random_number) mod 256
   If l mod 3 == 2:
   the ASCII code of scrambled_data[l] = the
      ASCII code of the processed_data[l] XOR
      random_number
}
```

Fig. 6. A pseudo-codes of the content substitution operation in the data-scramble function

After finishing the content substitution operation, the data-scramble function performs the location transposition operation. Also, the location transposition operation depends on the length of

```
scrambled_ctr = 0
for (i=0; i < the length of processed_data; i=i+1)
   {
    gen_loc_array[i] = False
}
while scrambled_ctr < the length of
processed_data
  {
   loc = random_number mod the length of
         processed_data
    if gen_loc_array[loc] == False
      {
       scramled_data[scrambled_ctr] =
        processed_data[loc]
       gen_loc_array[loc] = True
      }
   scrambled_ctr = scrambled_ctr + 1
}
```

Fig. 7. A pseudo-code of the location transposition function

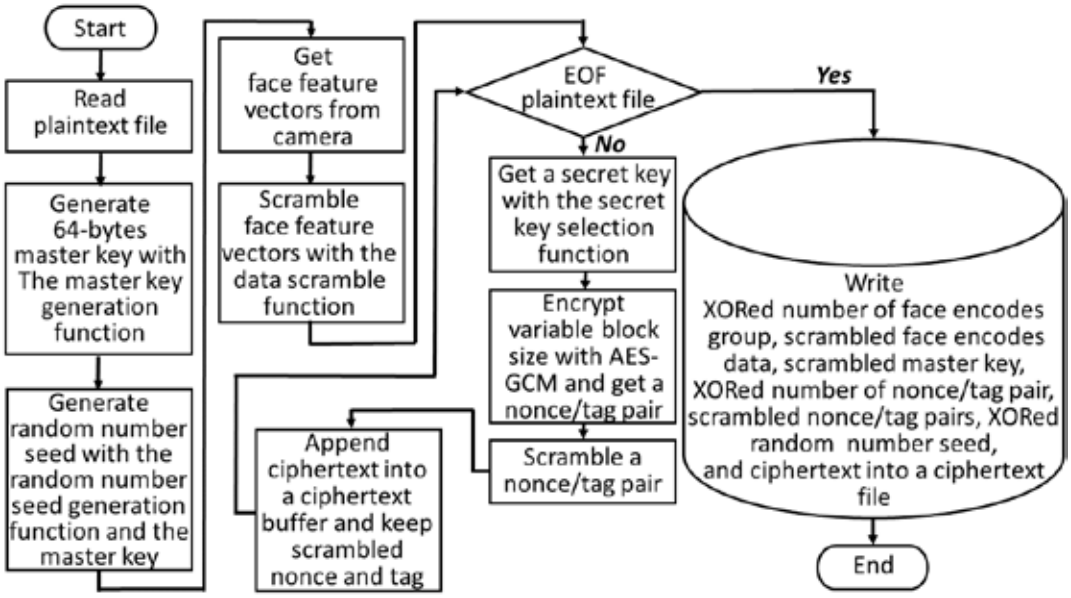

Fig. 8. A flowchart of the encryption process

the processed data to generate a series of random numbers as the location indexes of the scrambled data. Since it is possible to get the identical location index more than once by the generated random numbers during location transposition processes, the location transposition function uses a boolean array to record whether a specific location of scrambled data is swapped and avoids swapping one position of processed data repeatedly. Figure 7 shows the pseudo-code of the location transposition function.

The encryption process will encrypt a plaintext file with several functions mentioned in the above paragraphs; first, it opens a plaintext file and generates a master key corresponding to the plaintext file with the master-key generation function. And; the encryption process depends on the random number seed generation function to generate a random number seed with the generated master key for the following encryption process. Then, it gets the face feature vectors of an encryptor from a camera and scrambles the extracted face feature vectors with the data scramble function. Next, the encryption process performs an encryption process loop with the AES-GCM mechanism.

In the encryption process loop, the encryption process uses the secret-key selection function from the master key to get a secret key for the AES-GCM mechanism and determines a variable block size with a generated random number; then, it reads plaintext content from the opened plaintext file with the generated block size and has an encryption process with the AES-GCM mechanism. When the AES-GCM mechanism performs an encryption operation, it will produce a pair of nonce and tag for authentication in the decryption process. The encryption process uses the data scramble function to scramble the produced nonce and tag and keeps them. Also, the encryption process saves the ciphertext into a buffer temporarily. This encryption process loop does not end until the end of the plaintext file.

When the proposed encryption process finishes the encryption process loop job, it will store the XORed number of face encodes group, the scrambled face encodes data, the scrambled secret key set, the XORed number of nonce/tag pairs, the scrambled nonce/tag pairs, the XORed random number seed, and the ciphertext into a ciphertext file. Figure 8 shows the flowchart of the encryption process.

3.3 Decryption process

The decryption process performs a ciphertext decryption job with several functions. Those functions include a face recognition function, a master-key generation function, a secret-key selection function, and a data-unscramble function; only the data-unscramble

DOI：10.30188/JCCIT.202405_53(1).0006

function is a new function in the decryption process. Also, the data-unscramble function depends on applications of a pseudo-random number generation and an XOR operation.

- A data-unscramble function

Since the encryption process scrambles several critical data, such as the master key, face feature vectors, and pairs of nonce/tag, and stores them in the ciphertext file, the decryption process requires a function to unscramble those data in a readable pattern. The data-unscramble function is a reverse function of the data-scramble function in the encryption process; the data-scramble function uses the content substitution operation to substitute the content of plaintext data with unreadable data first; then, it performs the location transposition operation to swap locations of the plaintext data. Therefore, the data-unscramble function would depend on those two operations with some modifications to unscramble scrambled data. Figure 9 shows the pseudo-code of the data-unscramble function.

Fig. 9. A pseudo-code of the data-unscramble function

```
Step 1: to set random number seed
        for (i = 0; i < the length of the scrambled
             data; i = i + 1)
            {
             random_number_array[i] =
             random number
            }
Step 2: to swaps the scrambled data with the
        location transposition operation of the
        data-unscramble function and a series of
        generated random numbers
Step 3: to substitute the swapped scrambled data
        with the content substitution operation of
        the data-unscramble function and the
        random_number_array
```

First, the data-unscramble function depends on the random number seed used in the encryption process to generate a series of random numbers; then, it keeps those random numbers for a content substitution operation later. Next, the data-unscramble function performs a location transposition operation with the scrambled data and a series of generated random numbers. The location transposition operation is the same as the location transposition operation in the data-scramble function of the encryption process. After finishing the location transposition operation, the data-unscramble function performs a content substitution operation. The content substitution operation also depends on the ASCII code of scrambled data and a series of kept random numbers to unscramble the scrambled data with three simple arithmetic/logical operations, addition, subtraction, and XOR. However, those three simple arithmetic/logical operations have different execution sequences in the data-unscramble function. Figure 10 shows the pseudo-code of the content substitution operation in the data-unscramble function.

Fig. 10. A pseudo-code of the content substitution operation in the data-unscramble function

```
for (i =0; i < the length of processed data; i =
    i+1)
  {
  If i mod 3 == 0:
     the ASCII code of scrambled_data[i] = (the
     ASCII code of the processed_data[i] -
     random_number) mod 256
   If l mod 3 == 1:
     the ASCII code of scrambled_data[i] = (the
     ASCII code of the processed_data[i] +
     random_number) mod 256
   If l mod 3 == 2:
     the ASCII code of scrambled_datali] = the
     ASCII code of the processed_data[i] XOR
     random_number
}
```

For getting the critical parameters in the encryption process, the decryption process requires retrieving the XORed parameters, the number of face feature vectors, the number of nonce/tag groups, and the random number seed in the encryption process from a ciphertext file first. Then, it has XOR operations on those retrieved parameters with the specific XOR numbers to receive the values of those critical parameters. With those key parameters, the decryption process reads the scrambled face feature vectors from a ciphertext file and unscrambles them with the data-unscramble function and a series of random numbers. Next, the decryption process gets the face feature vectors of a decryptor from a camera; then, it has a comparison between these two face feature vectors. If the decryptor's face feature vectors are not equal to the face feature vectors stored in the ciphertext file, the decryption process shows a non-authority warning message; otherwise, the decryption process keeps doing the decryption process.

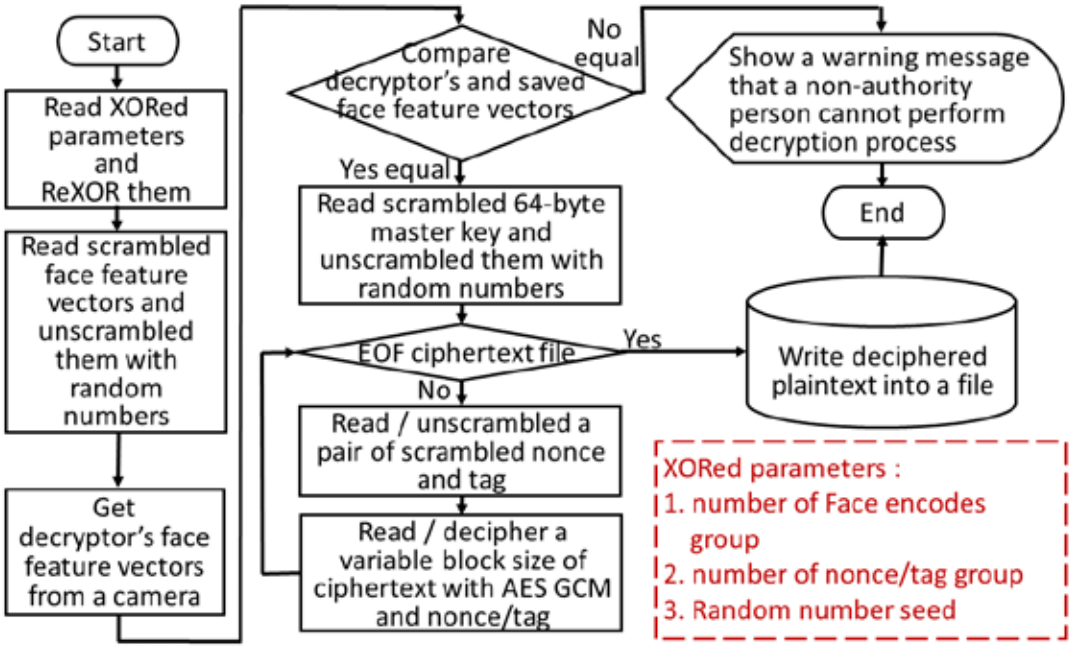

Fig. 11. A flowchart of the decryption process

When the decryption process begins to decrypt the ciphertext file, first, it reads the scrambled master key and all scrambled nonces and tags from the ciphertext file and performs an unscramble process with the data-unscramble function to get the correct master key and all unscrambled nonces/tags. Next, the decryption process depends on the number of the nonce/tag pairs to determine the number of a decryption loop. In each decryption loop, the decryption process selects a secret key from the master key randomly with generated random numbers and decrypts the ciphertext in a variable length with the AES-GCM mechanism and a pair of nonces and tags. Then, the decryption process saves the decrypting result into a buffer. As the decryption process decrypts all ciphertexts from a ciphertext file, it stores the decrypting result buffer into a plaintext file. Figure 11 shows the flowchart of the decryption process.

3.4 An exploration of the ciphertext file

The ciphertext file contains six kinds of XORed and scrambled data besides the ciphertext. The decryption process will depend on those unreadable data to decrypt the ciphertext. The first column of the unreadable data is the XORed number of the face features groups, and its length is 1 byte; it records that the encryption process uses how many groups of facial features. The second column of the unreadable data is the scrambled face feature data. Since a group of facial features contains 128 feature values and the length of one feature value is 48 bytes, the scrambled face feature data depends on the value of the number of the face features group to have a relatively variable length. The third column of the unreadable data is the scrambled master key; its length is 64 bytes. The fourth column of the unreadable data is the XORed number of the nonce/tag groups. The length of this column is 6 bytes; in this column, it records the number of nonce/tag groups generated by the encryption process. The fifth column of the unreadable data is the scrambled nonces and tags. Since the length of each pair of nonce and tag is 16 bytes, this column depends on the value of the number of nonce/tag groups to have a corresponding variable length. The sixth column of the unreadable data is the XORed random number seed; its length is 4 bytes. Figure 12 shows the format of a ciphertext file.

Further exploring the format of the ciphertext file, the length of the number of face feature groups is one byte; it allows the AEFR scheme to have 1~255 encryptors to perform an encryption/decryption process. Since the number

DOI：10.30188/JCCIT.202405_53(1).0006

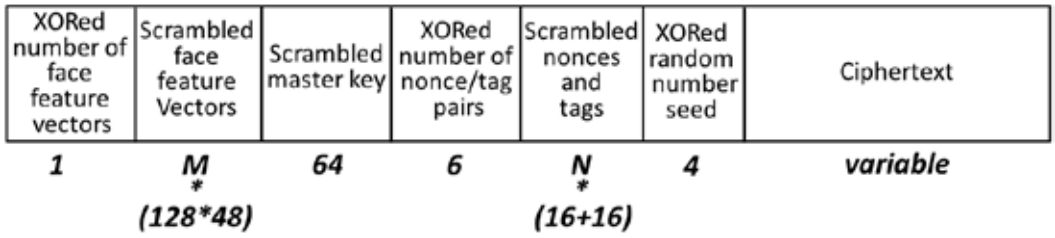

Fig. 12. A file format of the ciphertext file

of encryptors is dynamic, the length of the scrambled face feature data is variable, too. Then, it would cause the opponent cannot find the location of the scrambled master key precisely in the ciphertext file; this can cause the invisibility of the master key required by the encryption/decryption process further and reduce the possibility that the opponent compromises the ciphertext.

Moreover, the number of the nonce/tag pairs depends on an encryption job uses how many variable-size blocks. Since the encryption scheme uses a random number to generate a variable encryption block size in each AES-GCM encryption process, it is difficult for the AEFR scheme to predict how many nonce/tag pairs are yielded by the AES-GCM algorithm, especially for a pretty large plaintext file. Therefore, the AEFR scheme uses 6 bytes to store the number of nonce/tag pairs generated by the AES-GCM algorithm during the plaintext encryption job. The 6 bytes will store a pretty large integer; this could allow the AEFR scheme to handle an encryption/decryption job for an almighty big plaintext file.

The AEFR scheme uses a series of random numbers with different random number seeds to generate many variable-size blocks for a plaintext file in an encryption job. Each plaintext depends on its content to have a different number of variable-size block encryption processes; it is difficult to seize the number of variable-size blocks for a plaintext. Thus, this causes an opponent hard to know the length of nonce/tag pairs; then, he(she) cannot locate the store location of the XORed random number seed correctly in the ciphertext file.

The AEFR scheme is different from the traditional symmetric encryption mechanism; it depends on the face feature vectors of an encryptor to perform an encryption job without inputting a secret key by the encryptor and stores all information required to decrypt a ciphertext in a ciphertext file. For an encryption job, the face feature vectors of the encryptor are an initiator for the AEFR scheme to trigger the encryption process; for a decryption job, the face feature vectors of a decryptor are a threshold for the AEFR scheme to identify whether the decryptor is allowed to perform a decryption job or not.

Since all information required in a decryption job will store in a ciphertext file, the AEFR scheme scrambles or XORs the information into unreadable content. Moreover, the AEFR scheme depends on the master key to generate a secret key for each AES-GCM encryption operation; it also uses the random number seed to get a series of random numbers and scrambles the master key and the nonce/tag pairs with those random numbers. Therefore, the master key and the random number seed are the most critical information in a ciphertext file. The AEFR scheme improves the location invisibility of the scrambled master key and the XORed random number seed in a ciphertext file by arranging variable-length data before them separately.

IV. IMPLEMENTATION AND STUDY RESULTS

In this section, we explain how to implement the AEFR scheme on a personal computer first; then, we demonstrate the operation results of the AEFR scheme with a prototype program.

4.1 The AEFR scheme implementation

Since it is necessary for the AEFR scheme to recognize the face of an encryptor and a decryptor, this study implements a prototype program of the AEFR scheme on a Windows-based notebook computer equipped with a

camera. We code the AEFR scheme with Python and import several required modules to support the encryption/decryption operations in the AEFR scheme. Table 2 lists the information about the AEFR scheme's development platform.

Table 2. The AEFR scheme's development platform

Hardware		Software	
CPU	Intel i5	OS	Windows 11
RAM	8 GB	IDE	Pycharm
GPU	None	Python	3.10

The prototype program of the AEFR scheme imports several required modules, AES, hashlib, random, cv2, and face_recognition, to support the functionalities in the AEFR scheme. Those functionalities include the AES-GCM function, the SHA512 function, the random-number generation function, the camera control function, and the face recognition function. Also, this study codes several functions with Python to implement some functionalities, such as generating a random-number seed, scrambling/unscrambling secret key and nonce/tag pairs, converting face feature vectors into bytes, and writing/reading byte-based face feature vectors. The AEFR scheme's prototype program depends on the functions supported by the imported modules and the coded functions by ourselves to perform an encryption/decryption process for encryptors/decryptors.

4.2 Encryption/decryption process with the AEFR scheme's prototype program

When a user executes the AEFR scheme's prototype program, the prototype program lets users select a process, encryption, or decryption. When the user selects an encryption process, the prototype program asks the user to input a plaintext file name (see Figure 13); then, it invokes the encryption function to perform an encryption process. The encryption process depends on the content of the plaintext file to get the master key required in the encryption process with the SHA512 function first; then, it gets the face feature vectors of the encryptor from the camera of a notebook computer (see Figure 14). The encryption process encrypts plaintext into ciphertext with the AES-GCM mechanism and saves ciphertext, scrambled master key, scrambled encryptor's face feature vectors, scrambled nonce/tag pairs, and all XORed data in a ciphertext file with an extension "enc" (see Figure 15).

Figure 16 shows the execution result of the encryption process; it shows the encryption process encrypts the content of the plaintext file with several variable-size blocks. Figure 17 shows the contents of a plaintext file and the content of its corresponding ciphertext file. The ciphertext in Figure 17 is unreadable; this means the AEFR scheme's encryption process can encrypt plaintext into ciphertext. Moreover, examining the sizes of the plaintext file and the ciphertext file, the ciphertext file's size is larger than the plaintext file (see Figure 15); this shows the encryption process of the AEFR scheme stores some data required to decrypt into the ciphertext file.

When a user selects a decryption process with the prototype of the AEFR scheme, the prototype program asks the user to input a ciphertext file with an extension "enc" (see Figure 18). After inputting a ciphertext file name, the prototype program invokes the decryption function to perform a decryption process. The decryption process depends on the format of a ciphertext file to get the number of face feature vectors and the number of nonce/tag pair with XOR operations first; Then, it can correctly get the random number seed with another XOR operation from the ciphertext file and unscramble the master key and nonce/tag pairs. When all data used for the decryption process is ready, the decryption process invokes the notebook computer's camera to get the face feature vectors of the decryptor (see Figure 19). The decryption process compares the face feature vectors from the camera with the face feature vectors stored in the ciphertext file. If the comparison result is positive, the decryption process keeps decrypting the ciphertext file; otherwise, the decryption process would show a message that the person in front of the camera has not the privilege to decipher the ciphertext file (see Figure 20).

DOI：10.30188/JCCIT.202405_53(1).0006

Fig. 13. The screen of an encryption process selection and a plaintext file input for the AEFR scheme program

Fig. 14. A capture of the face feature vectors of an encryptor with a camera

Fig. 15. The ciphertext file with “enc” extension

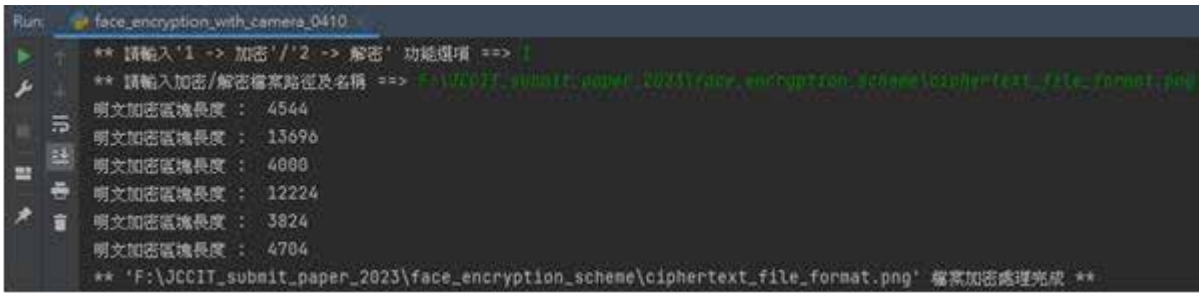

Fig. 16. The execution result of the encryption process

Figure 16 shows the execution result of the encryption process; it shows the encryption process encrypts the content of the plaintext file with several variable-size blocks. Figure 17 shows the contents of a plaintext file and the content of its corresponding ciphertext file. The ciphertext in Figure 17 is unreadable; this means the AEFR scheme's encryption process can encrypt plaintext into ciphertext. Moreover, examining the sizes of the plaintext file and the ciphertext file, the ciphertext file's size is larger than the plaintext file (see Figure 15); this shows the encryption process of the AEFR scheme stores some data required to decrypt into the ciphertext file.

When a user selects a decryption process

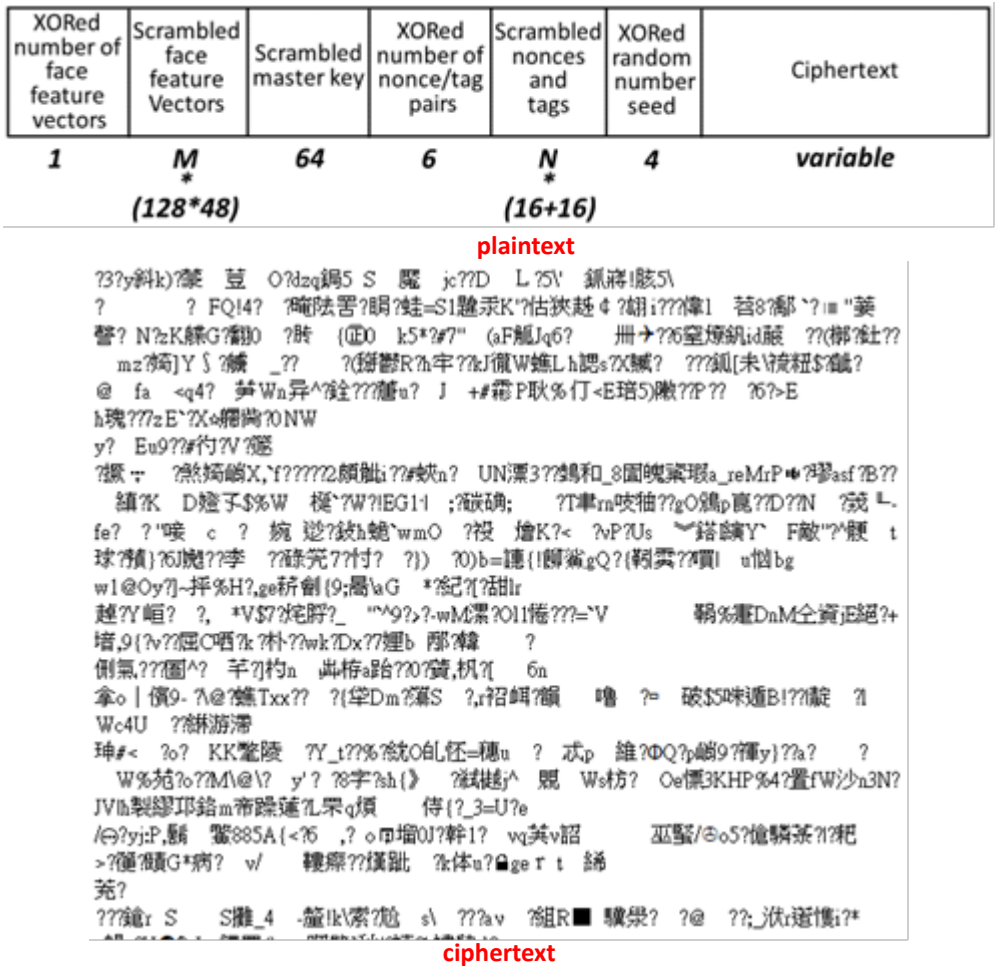

Fig. 17. The contents of the plaintext file and the ciphertext file

with the prototype of the AEFR scheme, the prototype program asks the user to input a ciphertext file with an extension "enc" (see Figure 18). After inputting a ciphertext file name, the prototype program invokes the decryption function to perform a decryption process. The decryption process depends on the format of a ciphertext file to get the number of face feature vectors and the number of nonce/tag pair with XOR operations first; Then, it can correctly get the random number seed with another XOR operation from the ciphertext file and unscramble the master key and nonce/tag pairs.

When all data used for the decryption process is ready, the decryption process invokes the notebook computer's camera to get the face feature vectors of the decryptor (see Figure 19). The decryption process compares the face feature vectors from the camera with the face feature vectors stored in the ciphertext file. If the comparison result is positive, the decryption process keeps decrypting the ciphertext file; otherwise, the decryption process would show a message that the person in front of the camera has not the privilege to decipher the ciphertext file (see Figure 20).

When the decryption process completes the decryption job of the ciphertext file (see Figure 21), it saves the deciphered plaintext into a file with an 'R' extension (see Figure 22). Looking at

Fig. 18. The screen of a decryption process selection and a ciphertext file input in the prototype program of the AEFR scheme

DOI：10.30188/JCCIT.202405_53(1).0006

Fig. 19. A capture of the face feature vectors of a decryptor with a camera

Fig. 20. An unauthorized message to perform a decryption process

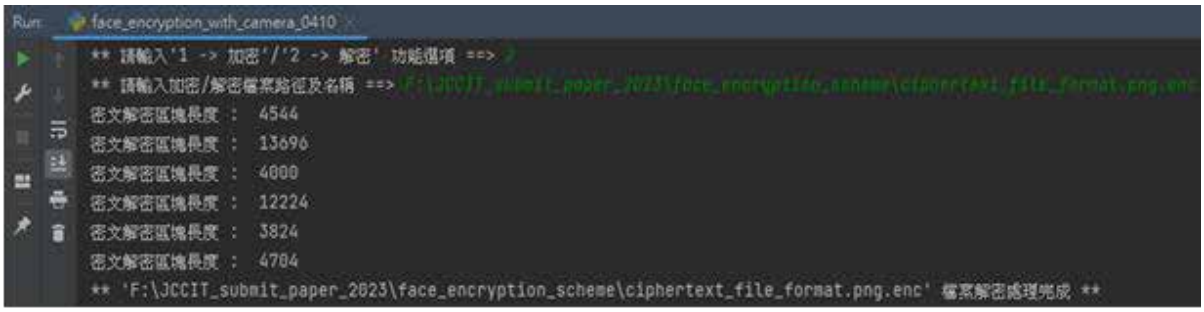

Fig. 21. The execution result of the decryption process

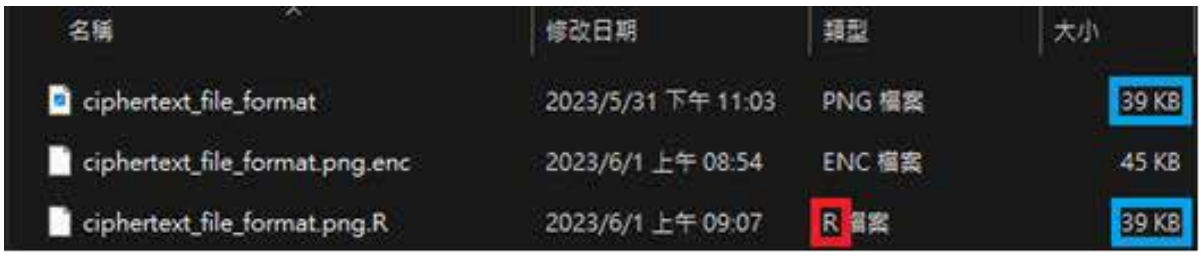

Fig. 22. The deciphered plaintext file with “R” extension

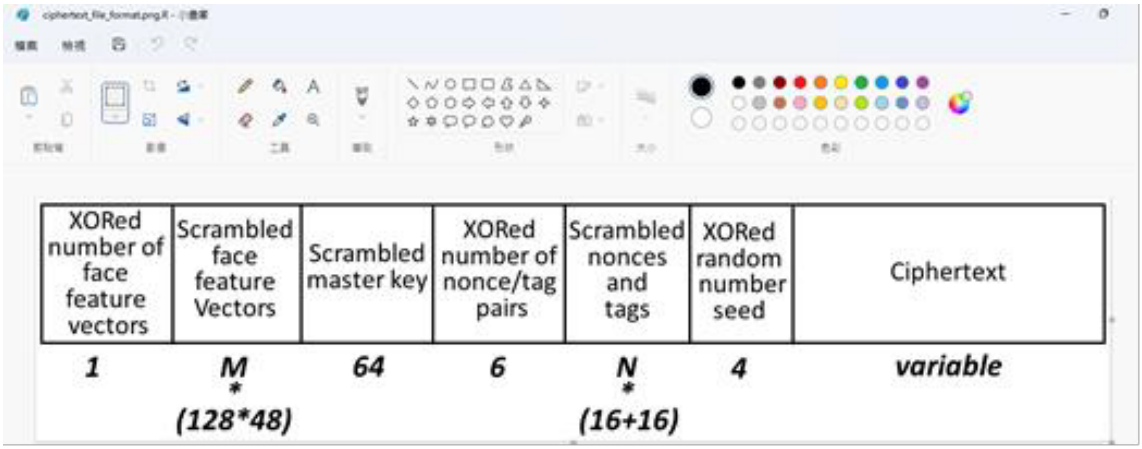

Fig. 23. The content of the decrypted plaintext file

the decryption process result in Figure 21, we can find that the decryption process deciphers the ciphertext with several variable-size blocks corresponding to the variable-size blocks in the encryption process. Figure 23 shows the content of the decrypted plaintext file; it can display the format of a ciphertext file. It means that the prototype program of the AEFR scheme can decipher a ciphertext file correctly.

4.3 A validation of the encryption/ decryption process of the AEFR scheme

This study tries to realize the ability that the AEFR scheme's prototype program to encrypt/decrypt different types of files; we use four types of files, txt, png, pdf, and pptx, to execute encryption/decryption processes with the prototype program of the AEFR scheme and tries to open those deciphered plaintext files with the proper software. The results show that the software can open those deciphered plaintext files normally; this means the AEFR scheme's prototype program can decrypt the ciphertext files correctly.

Also, this study depends on the original plaintext file and the deciphered plaintext file to validate the encryption/decryption correction of the AEFR scheme's prototype program. This study uses two ways to compare the contents of the original plaintext file and the deciphered plaintext file to validate the content correctness of the decrypted plaintext file. One is the Windows "comp" command; Figure 24 shows the comparison results of those files. The comparison results show that all the deciphered plaintext files are the original-plaintext files. And; the other is to use the SHA256 algorithm to validate the sameness of file content. Table 3 shows all the SHA256 message digests of the original/deciphered plaintext files. Looking at Table 3, it is clear that each pair of the original/deciphered plaintext files receive the same SHA256 message digest. Therefore, the contents of each of the original/deciphered

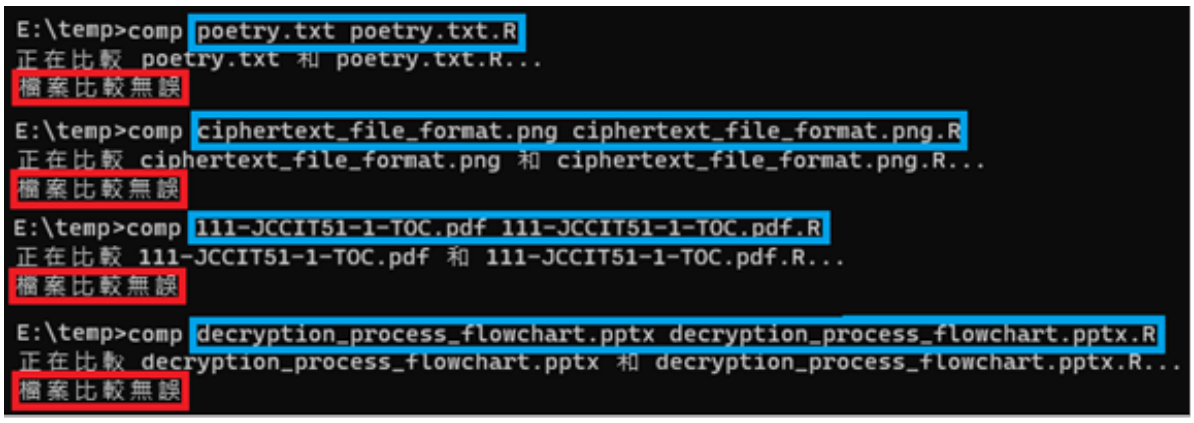

Fig. 24. The Windows "comp" command comparison result

DOI：10.30188/JCCIT.202405_53(1).0006

plaintext files are the same. These two comparison results demonstrate that the AEFR scheme's prototype program can decrypt the encrypted file correctly.

Table 3. The SHA256 message digest comparison results

File name	SHA256 message digest
P	e25b7cb96db573367f3838d5eb6955f4a25d1c5a825786a4c8d77d787d63c3fb
P.R	e25b7cb96db573367f3838d5eb6955f4a25d1c5a825786a4c8d77d787d63c3fb
C	786233fd2106cf32ea0240632dad66c03257e555f5ad51cda137d8168155030e
C.R	786233fd2106cf32ea0240632dad66c03257e555f5ad51cda137d8168155030e
1	87333df477d75af27039f0107693636d0bfcd35306d24fcbc72cd00caf73233e
1.R	87333df477d75af27039f0107693636d0bfcd35306d24fcbc72cd00caf73233e
D	f5c476f8a2d15593bdad3bf6d3e490a47f6500e84bd90db099354e5205d363ab
D.R	f5c476f8a2d15593bdad3bf6d3e490a47f6500e84bd90db099354e5205d363ab
Legend: P: poetry.txt, C: ciphertext_file_format.png, 1: 111-JCCIT51-1-TOC.pdf, D: decryption_process_flowchart.pptx . R: ciphertext recovery file	

Table 4 shows the statistics of variable block size for each plaintext file. Data in Table 4 demonstrates that the AEFR scheme's prototype program performs an encryption job for each plaintext file with different variable-size blocks. Also, the size of each block is different from the others. Usually, the larger a plaintext file size is, the more variable-size blocks have. Since the AEFR scheme's prototype program depends on a series of random numbers to assign all encrypted block sizes, thus, it is difficult to predict the size of a plaintext block; this would reduce the possibility that the opponent compromises the ciphertext file.

Table 4. The statistics of encryption variable block size of plaintext files

File name	File length (byte)	Variable block size (byte)	Block number
P	2745	2745	1
C	39460	4544, 13696, 4000, 12224, 3824, 1172	6
1	245740	7744, 8112, 6032, 4480, 15664, 3056, 3808, 3120, 13488, 10720, 432, 6736, 5584, 11120, 11792, 992, 10640, 12576, 9696, 15440, 13920, 176, 7648, 1456, 10416, 9200, 1856, 12608, 3920, 11264, 8816, 3228	32
D	50569	13952, 7072, 14112, 11664, 3769	5
Legend: P: poetry.txt, C: ciphertext_file_format.png, 1: 111-JCCIT51-1-TOC.pdf, D: decryption_process_flowchart.pptx			

V. CONCLUSION

As face-recognition technology becomes mature, many face-recognition applications keep emerging. This study proposes an AES-GCM-based symmetric block encryption scheme with face-recognition technology (the AEFR scheme). The AEFR scheme uses the SHA512 message digest as a master key for the encryption/decryption process without inputting a secret key; it also depends on the master key to generate a random number seed. The AEFR scheme performs an encryption process of a plaintext file with the master key and the generated random number seed; it stores all data used in the decryption job in the ciphertext file.

Moreover, the AEFR scheme depends on a face-recognition result to determine whether one person in front of a camera can decrypt a ciphertext file. When the AEFR scheme performs the decryption process of a ciphertext file, it uses face recognition technology to confirm the identification of the decryptor. If

one person in front of a camera is the person who has the privilege to perform a decryption job, the AEFR scheme performs the decryption process; otherwise, the AEFR scheme shows a warning message.

This study implements the prototype program of the AEFR scheme with Python. We use four types of files, txt, png, pdf, and pptx, to validate the encryption/decryption correctness of the prototype program of the AEFR scheme. The results show that the prototype program can correctly decipher all ciphertext files; it confirms the feasibility and practicality of the AEFR encryption scheme. Therefore, a person can depend on his (or her) face to invoke the AEFR encryption scheme and execute an encryption/decryption job without a secret key input; this can avoid the issue of a secret key keeping in an encryption mechanism.

The AEFR encryption scheme is an AES-based encryption scheme with a face recognition mechanism; therefore, its time/space complexity depends on the AES. The AEFR encryption scheme randomly selects 32 bytes (256 bits) from the master-key as a secret key in each encryption/decryption round; thus, it can resist a key-recovery attack with a total complexity of 2^{131} time and 2^{65} memory.

REFERENCES

[1] Barnouti, Nawaf Hazim, Al-Dabbagh, Sinan Sameer Mahmood, and Matti, Wael Esam. "Face recognition: A literature review." *International Journal of Applied Information Systems* 11.4 (2016): 21-31.

[2] Jain, Anil K. and Li, Stan Z. *Handbook of face recognition*. Vol. 1. New York: springer, 2011.

[3] Zhao, Wenyi, et al. "Face recognition: A literature survey." ACM computing surveys (CSUR) 35.4 (2003): 399-458.

[4] Hjelmås, Erik, and Low, Boon Kee. "Face detection: A survey." Computer vision and image understanding 83.3 (2001): 236-274.

[5] Brimblecombe, Phil. "Face detection using neural networks." H615–Meng Electronic Engineering, School of Electronics and Physical Sciences, URN 1046063 (2002).

[6] Kaur, Rajkiran, and Rajput, Rachna. "Face recognition and its various techniques: a review." International journal of scientific engineering and technology research 2.3 (2013): 670-675.

[7] Bruce, Vicki, and Young, Andy. "Understanding face recognition." British journal of psychology 77.3 (1986): 305-327.

[8] Himanshu, S. Dhawan, and Neha Khurana. "A Review of Face Recognition."

[9] International journal of research in engineering and applied sciences 2.2 (2012): 921-939.

[10] Ellis, H. D. "Introduction to aspects of face processing: Ten questions in need of answers." Aspects of face processing (1986): 3-13.

[11] Bakshi, Urvashi, and Singhal, Rohit. "A survey on face detection methods and feature extraction techniques of face recognition." International Journal of Emerging Trends & Technology in Computer Science (IJETTCS) 3.3 (2014): 233-237.

[12] Penard, Wouter, and Werkhoven, Tim van. "On the secure hash algorithm family." Cryptography in context (2008): 1-18.

[13] Guesmi, Ramzi, et al. "A novel chaos-based image encryption using DNA sequence operation and Secure Hash Algorithm SHA-2." Nonlinear Dynamics 83 (2016): 1123-1136.

[14] Pittalia, Prashant P. "A comparative study of hash algorithms in cryptography." International Journal of Computer Science and Mobile Computing 8.6 (2019): 147-152.

[15] Sklavos, Nicolas, and Koufopavlou, Odysseas. "On the hardware implementations of the SHA-2 (256, 384, 512) hash functions." Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.. Vol. 5. IEEE, 2003.

[16] Pub, NIST FIPS. "180-2,“Announcing the Secure Hash Standard,” Aug. 1, 2002, pp. i-iii and 3-16." Available from NTIS at above address.

[17] Dworkin, Morris J. "SHA-3 standard: Permutation-based hash and extendable-output functions." (2015).

[18] Daemen, Joan, and Rijmen, Vincent. "AES proposal: Rijndael." (1999).

[19] FIPS, PUB. "197: Specification for the Advanced Encryption Standard (AES)." Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD (2001): 20899-8900.

DOI：10.30188/JCCIT.202405_53(1).0006

[20] Dobbertin, Hans, Knudsen, Lars, and Robshaw, Matt. "The cryptanalysis of the AES–a brief survey." International Conference on Advanced Encryption Standard. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[21] Biryukov, Alex, Khovratovich, Dmitry, and Nikolić, Ivica. "Distinguisher and related-key attack on the full AES-256." Annual International Cryptology Conference. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[22] Bogdanov, Andrey, et al. "Bicliques with minimal data and time complexity for AES." Information Security and Cryptology-ICISC 2014: 17th International Conference, Seoul, South Korea, December 3-5, 2014, Revised Selected Papers 17. Springer International Publishing, 2015.

[23] Housley, Russell. "Using AES-CCM and AES-GCM authenticated encryption in the cryptographic message syntax (CMS)." No. rfc5084. 2007.

[24] Krovetz, Ted, and Rogaway, Phillip. "The software performance of authenticated-encryption modes." Fast Software Encryption: 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers 18. Springer Berlin Heidelberg, 2011.

[25] Dworkin, Morris J. "Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac." (2007).

[26] Venkatesan, R., et al. "Secure online payment through facial recognition and proxy detection with the help of TripleDES encryption." Journal of Discrete Mathematical Sciences and Cryptography 24.8 (2021): 2195-2205.

[27] Ameen, Zinah Jaffar Mohammed. "Face Recognition Integrated with Chaotic Encryption for Secure Electronic Election Application." Multi-Knowledge Electronic Comprehensive Journal For Education And Science Publications 23 (2019): 1-19.

[28] Battaglia, F., Iannizzotto, Giancarlo, and Bello, L. Lo. "A biometric authentication system based on face recognition and rfid tags." Mondo Digitale 13.49 (2014): 340-346.

[29] Sawant, Vedant A., et al. "Face Recognition Based Password Encryption and Decryption System." 2023 4th International Conference for Emerging Technology (INCET). IEEE, 2023.

[30] Vankadara, Anurag, et al. "Enhancing Encryption Mechanisms using SHA-512 for user Authentication through Password & Face Recognition." 2023 International Conference on Inventive Computation Technologies (ICICT). IEEE, 2023.

[31] Chandrasekhar, Tadi, and Kumar, Sumanth. "A noval method for cloud security and privacy using homomorphic encryption based on facial key templates." Journal of Advances in Information Technology 13.6 (2022).